TECNOLÓGICO DE MONTERREY®

Campus Ciudad de México

Escuela de Graduados en Ingeniería y Arquitectura

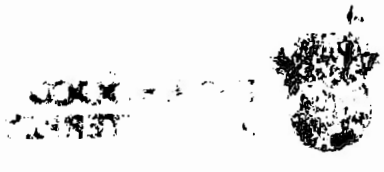
Maestría en Ciencias de la Computación

"Modelo de visualización en internet de espacios de estados finitos de una red reguladora genética de más de mil nodos"

Autor:
Marcos Alberto Bernal Castillo

Directores de tesis:
Dra. María Alicia Aviñó Díaz
Dr. César Augusto Coutiño Gómez

Abril, 2012



"Modelo de visualización en internet de espacios de estados finitos de una red reguladora genética de más de mil nodos"

por

Marcos Alberto Bernal Castillo

ha sido aprobada

abril 2012

por la comisión de tesis:

Dr. Benjamín Hernández Arreguín _____ , Presidente

Dra. María Alicia Aviñó Díaz _____ , Sinodal

Dr. César Augusto Coutiño Gómez _____ , Sinodal

Aceptada

_____

Dr. José Martín Molina Espinosa

Agradecimientos

Este trabajo no hubiera sido posible sin el apoyo constante de mi familia quien me dio apoyo moral incondicional y consejos, mis amigos quienes siempre me impulsaron a continuar hasta el final y a mis compañeros de maestría y trabajos quienes me apoyaron moralmente.

En especial quiero agradecer a mi madre Guadalupe Castillo Pacheco por su invaluable apoyo en cualquier ámbito durante la realización de esta tesis, desde apoyo moral, trámites y cuidados especiales, hasta revisiones de tesis y proyecto.

A mi hermana Claudia Berenice Bernal Castillo y su hija Sara Chalé Bernal por apoyarme en momentos importantes y darme espacios para escribir la tesis.

A mi padre Marco Antonio Bernal Montaño por brindarme oportunidades y guiarme con sus consejos.

A mi abuela María del Pilar Pacheco por apoyarme en los días de trabajo.

A mi director de tesis, César Augusto Coutiño Gómez por impulsarme a estudiar la maestría y guiarme con el tema de tesis y sus valiosos consejos académicos y personales.

A mi directora de tesis, María Alicia Aviñó Díaz por tener la paciencia de guiarme por el tema de bioinformática y por sus consejos y apoyos durante el desarrollo del proyecto y de la tesis.

A Jésica Zermeño Núñez por sus valiosas ediciones del documento de tesis y apoyo moral.

A Rosbel Serrano Torres y Emmanuel Ramírez González por su apoyo para crear secciones especiales con código y formatos especiales además del apoyo moral.

A Ken Gibel por abrir la oficina un viernes por la noche en un momento crucial del desarrollo.

A mis jefes Sairám Cerón Alegre, Mario Aliaga Valenzuela, Ultiminio Ramos Galán y Ziad Chamma por apoyarme con tiempos y permisos para la realización de esta tesis.

Resumen

Como parte del estudio genómico, se encuentra el análisis por microarreglos, el cual busca obtener datos de las circunstancias de los genes con la intención de descubrir el comportamiento de los genes que alteran las células y tejidos de enfermedades letales, principalmente. La información que se obtiene de los microarreglos es analizada, normalizada y adaptada a diferentes modelos matemáticos. Uno de los retos que se presentan es la vasta cantidad de información generada y el procesamiento computacional que permita un análisis especializado y bien informado.

El proyecto que se presenta busca apoyar este análisis a partir de los datos que introduzca la persona especializada y generar redes visualmente comprensibles donde sobresalga de manera gráfica la relación entre grupos de genes, su comportamiento y, más importante, los estados estacionarios dentro de la red. Los estados estacionarios indicarán cuándo el grupo de genes se ha establecido, posiblemente indicando cuándo el tejido ha muerto.

Se desarrolló un modelo visual disponible al publico en internet que toma las entradas, analiza la sintaxis de los parámetros de entrada, genera la red (grafo) lógicamente, identifica los estados, resalta los estados estacionarios, genera una tabla de estadísticas y agrega otras herramientas para su mejor análisis, como búsqueda de nodos, aumento y disminución de tamaño, impresión de nodos, maximización del espacio visual y tutorial gráfico.

Con el desarrollo de más herramientas tecnológicas para el análisis de datos genómicos crece la visión para comprender nuestra esencia como seres vivos y la motivación por encontrar los patrones que conducen los procesos biológicos que nos hacen vivir, y vivir mejor.

Modelo de visualización en internet de espacios de estados finitos de una red reguladora genética de más de mil nodos

Contenido

LISTA DE TABLAS ..... **III**

LISTA DE FIGURAS ..... **IV**

CAPÍTULO 1. INTRODUCCIÓN **1**

1.1 ANTECEDENTES 1

1.2 PROCESO DEL ESTUDIO CON MICROARREGLOS 1

1.2.1 Análisis de genes *5*

1.2.2 Microarreglos *6*

1.3 PLANTEAMIENTO DEL PROBLEMA 8

1.3.1 Hipótesis *11*

1.4 OBJETIVO 12

1.5 RESULTADOS ESPERADOS 12

1.6 JUSTIFICACIÓN DE LA INVESTIGACIÓN 13

1.7 ALCANCE 14

1.8 ESTRUCTURA DEL DOCUMENTO 15

CAPÍTULO 2. MODELOS DE ANÁLISIS DE DATOS BIOLÓGICOS **17**

2.1 ANÁLISIS DE REDES REGULADORAS GENÉTICAS 17

2.2 SOFTWARE 19

CAPÍTULO 3. DESARROLLO DEL MODELO **22**

3.1 MATERIALES Y MÉTODOS 22

3.2 ANÁLISIS DEL MODELO 23

3.2.1 Requisitos funcionales *24*

3.2.2 Requisitos No funcionales *29*

3.2.3 Flujo del modelo *30*

3.2.4 Estructura del modelo *31*

3.3 DISEÑO DEL MODELO 32

3.3.1 Complejidades computacionales *33*

3.3.2 Proyecto Gefundus *37*

3.3.3 Proyecto Genodus *40*

3.3.4 Tecnologías utilizadas *43*

3.4 DESARROLLO DEL MODELO 48

3.4.1 Desarrollo de Gefundus. *48*

3.5 ALGORITMO DE RECORRIDO 50

3.5.1 Algoritmo de composición de los grafos. *54*

3.5.2 Algoritmo de creación de documento *61*

3.6 INTERFAZ GRÁFICA 67

3.6.1 Analizador sintáctico y semántico *71*

3.6.2 Dibujar Nodos *79*

3.6.3 Dibujar Curvas *82*

CAPÍTULO 4. VALIDACIONES **92**

4.1 PRUEBAS FUNCIONALES 93

4.1.1 Pruebas en condiciones normales *93*

4.1.2 Pruebas de límite *94*

4.2 PRUEBAS NO FUNCIONALES 94

4.2.1 Rendimiento *94*

4.2.2 Visualización *95*

4.2.3 Facilidad de navegación y usabilidad *96*

4.2.4 Multiplataforma y disponibilidad *96*

4.3 COMPARACIÓN CON SOFTWARE EXISTENTE Y FUNCIONES SIMILARES 97

4.3.1 Función principal de los sistemas *98*

4.3.2 Parámetros de entrada *98*

4.3.3 Presentación de los resultados. *98*
4.3.4 Rapidez de procesamiento. *98*
4.3.5 Límite de procesamiento. *99*
4.3.6 Construcción del sistema. *99*
4.4 EJEMPLO CON DATOS BIOLÓGICOS 101
CAPÍTULO 5. CONCLUSIONES 102
BIBLIOGRAFÍA 106

Lista de Tablas

Tabla 3-1 Definición del caso de uso Llenar formulario de entrada.....25
Tabla 3-2 Definición de caso de uso Generar espacio de estados 27
Tabla 3-3 Definición de caso de uso Visualizar espacio de estados.....28
Tabla 3-4 Definición de caso de uso Analizar espacio de estados.....28
Tabla 3-5 Características importantes para posicionar un elemento gráficamente.....55

Lista de Figuras

Figura 3.1 Diagrama UML de casos de uso del modelo 24
Figura 3.2 Diagrama UML de actividad del modelo 30
Figura 3.3 Diagrama UML de clases 32
Figura 3.4 Diagrama UML de actividad para generar espacio de estados 34
Figura 3.5 Diagrama UML de actividad de Gefundus 37
Figura 3.6 Diagrama UML de máquina de estados de Gefundus 38
Figura 3.7 Diagrama UML de clases de Gefundus 39
Figura 3.8 Diagrama UML de actividad de Genodus 40
Figura 3.9 Diagrama UML de máquina de estados de Genodus 41
Figura 3.10 Diagrama UML de clases de Genodus 42
Figura 3.11 Diagrama UML de despliegue del modelo 44
Figura 3.12 Uso de plataformas de software en internet de mercados maduros 46
Figura 3.13 Organización de la interfaz en *Flex* 68
Figura 3.14 Parámetros de entrada de Genodus 69
Figura 3.15 Funciones de entrada en Genodus 69
Figura 3.16 Regulación entre nodos por medio de funciones 70
Figura 3.17 Botón de envío de parámetros y procesamiento de Genodus 75
Figura 3.18 Tabla de estadísticas en Genodus 79
Figura 3.19 Ciclo de un solo nodo 82
Figura 3.20 Ciclo de dos o más nodos 83
Figura 3.21 Espacio de estados creado por Genodus y Gefundus 84
Figura 3.22 Herramienta de aumento y disminución de tamaño de Genodus 85
Figura 3.23 Herramienta de búsqueda de nodo de Genodus 86
Figura 3.24 Barra plegable para maximización de espacio visual de Genodus 87
Figura 3.25 Tutorial de Genodus, Introducción 89
Figura 3.26 Tutorial de Genodus, Número de elementos por nodo 89
Figura 3.27 Tutorial de Genodus, Número de estados 90
Figura 3.28 Tutorial de Genodus, Funciones de entrada 91
Figura 4.1 Tabla de desempeño de Genodus y Gefundus en memoria y tiempo de procesamiento 95
Figura 4.2 Genodus sobre un teléfono inteligente 97
Figura 4.3 Regulación del metabolismo de lactosa en E. Coli 101

CAPÍTULO 1

Introducción

1.1 Antecedentes

El estudio de la composición primordial de los seres vivos ha sido por años el foco de estudio de los biólogos, y tras el descubrimiento de la secuencia genética del Ácido Desoxirribonucleico (ADN), que representa el código del cual están hechos los seres vivos. Los avances en la ciencia hacia el estudio del genoma humano se multiplican cada día para intentar comprender la composición de las personas, animales y plantas con el objetivo de curar enfermedades o conocer hasta qué grado se puede alterar o no ese código para obtener resultados esperados. Uno de los objetivos más importantes de este estudio en la actualidad es el de conocer cómo se relacionan los genes entre si y cómo cambian estas relaciones cuando se altera su proceso normal de desarrollo.

1.2 Proceso del estudio con microarreglos

La genética es la ciencia que estudia los genes, ya que en ellos están las características de la herencia. Los genes se forman de segmentos de Ácido Desoxirribonucleico (ADN), el cual es una doble hélice consistente de dos cadenas interrelacionadas y complementarias de nucleótidos. El conjunto entero del ADN es el genoma del organismo. El ADN controla la estructura, la función y el comportamiento de las células y puede crear copias exactas, o casi, de sí mismo. Las moléculas de ADN están ensambladas en cromosomas, y los genes son la región funcional del ADN. En la historia de la genética, la herencia y la variación han constituido la base de la misma. En la investigación moderna, la genética proporciona herramientas importantes para la investigación de la función de los genes particulares, así como del análisis de interacciones genéticas. Los genes contienen la información necesaria

para determinar la secuencia de aminoácidos de las proteínas. Éstas, a su vez, desempeñan la función importante en la determinación del fenotipo final, o apariencia física, del organismo.

Durante años el estudio de los genes fue realizado de una manera muy lenta debido a la dificultad de medir cómo se altera un gen cuando alteramos otros genes o cuando alteramos su medio de existencia. A finales de los noventa, se construyó un aparato que permite medir la intensidad de una gran cantidad de genes en un solo experimento. Este aparato es conocido como microarreglo o *genechip*. Esto revolucionó el estudio de los genes y en la actualidad existen una gran cantidad de datos en internet que permiten estudiar y analizar las relaciones entre los genes, es decir, las Redes Genéticas. En esta tesis se presenta un software que permite visualizar los cambios que se realizan en los genes en el transcurso del tiempo en una Red Genética de estudio. Para ello se utiliza una función matemática que representa el cambio de estado en los genes y que, utilizando probabilidades permite ver los posibles cambios de estado y los puntos de equilibrio.

Con el experimento por microarreglos se obtienen datos de la combinación y el efecto de ciertos genes sobre los genes originales, es decir, un tejido que contiene genes sanos (originales), se combina con otro tejido que se conforma por genes afectados por alguna enfermedad como el cáncer, para ver el comportamiento en diferentes momentos y estudiar los cambios de los genes. Un microarreglo presenta datos numéricos de esta alteración.

Gracias a los estudios con microarreglos es posible:

- Estudiar miles de genes simultáneamente
- Generar una cantidad muy grande de datos para cierto estado
- Obtener diferentes estados de los genes a los que llaman análisis de expresión del gen.

Estas expresiones se van generando a través de tomar diferentes muestras para el microarreglo en diferentes etapas de vida de los tejidos combinados. Las expresiones indicarán qué genes están activados en cada etapa y de esta forma poder decir qué genes regulan a otros para que el microarreglo se encuentre en dicho estado.

Las redes reguladoras genéticas se refieren al uso de microarreglos para ver el comportamiento de un grupo de genes en diferentes puntos en el tiempo o incluso de genes que se afectan de manera simultánea. De esta forma, los biólogos pueden ver el comportamiento para intentar definir la estructura del sistema, buscando patrones que se relacionen con los datos generados.

Por medio de la ingeniería inversa es que se intenta encontrar el modelo que describa el comportamiento de la red reguladora de genes. Para hacer esto posible es necesario contar con una cantidad suficiente de datos generados correctamente por el uso de microarreglos. La mayoría de los patrones que se describen generarán funciones que representen el comportamiento o la regulación de los genes de un estado a otro.

La gran cantidad de datos generados por este estudio requiere un análisis que no puede ser resuelto fácilmente por métodos convencionales o en un tiempo razonable. Es por eso que se ha introducido el uso de modelos matemáticos, estadística y computación, para la facilitación el estudio, la modelación de los datos y el tiempo con la capacidad de realizar operaciones automatizadas.

La combinación de estas áreas de estudio compone el término de *Bioinformática* que mezcla los estudios biológicos con los recursos informáticos y matemáticos, con el fin de analizar, principalmente para analizar datos relacionados con el código genético.

Por la parte de matemáticas se involucran diferentes modelos, como son de ecuaciones diferenciales, bayesianos, probabilísticos, entre otros. Entre los más conocidos se encuentran el modelo booleano, para representar dos estados en los genes y el modelo de redes probabilísticas para más estados. El modelo PBN (por sus siglas en inglés, Probabilistic Boolean Network) se ha presentado por Shmulevich y E. Dougherty para estudiar el comportamiento de las redes reguladoras genéticas introduciendo probabilidades a la red, logrando así producir redes que permitan la predicción del estado de un gen a partir del estado de otros o el conocimiento del estado de equilibrio en un espacio de estados.

Para la aplicación de este tipo de modelo probabilístico es necesario que se desarrolle un algoritmo que cumpla con la generación de espacios de estados y los presente de forma gráfica, el cual le permita al biólogo generar conclusiones a partir de las relaciones entre expresiones de genes.

Existen en el mercado herramientas de software dedicadas a analizar este tipo de modelos, pero un aspecto importante a resaltar es que, con el uso de este modelo y con la cantidad de información prevista a ser analizada, es poco práctico que se intente resolver por una sola computadora en un tiempo aceptable ya que los datos generados para más de cien genes (tomando en cuenta que el microarreglo puede producir información para miles de genes) y las funciones que se tienen que aplicar a los datos de entrada tomarían mucho tiempo en ser analizadas.

Existen diferentes opciones conocidas en las ciencias computacionales para procesar gran cantidad de datos por medio del cómputo en paralelo. El cómputo en paralelo es un conjunto de procesadores que son capaces de trabajar cooperativamente para resolver un problema computacional. Esta definición abarca lo suficiente para incluir supercomputadoras en paralelo con cientos o miles de procesadores, redes de computadoras, computadoras con

procesadores múltiples, y sistemas embebidos. El cómputo en paralelo ofrece el potencial de concentrar recursos computacionales, sean procesadores, memoria o entrada/salida de ancho de banda, en problemas computacionales importantes [1]. Entre las opciones que existen se encuentra el procesamiento por hardware como múltiples procesadores como GPU[1], múltiples memorias o redes interconectadas, el procesamiento por Software como sistemas operativos en paralelo o construcciones para orquestrar concurrencia y aplicaciones de software como algoritmos paralelos.

El diseño de programas paralelos incluye la etapa de particionamiento que exponga las oportunidades de la ejecución en paralelo. Algunos de los retos que se pueden tener es la división de la información en partes iguales para ser procesada sin afectar la estructura de la información global y la realización gráfica.

Actualmente se presentan varias soluciones a estos retos, cubriendo algunos en cantidad de nodos que se procesan y otros en la forma visual para representarlos. En el Capítulo 2 se presentan soluciones que actualmente trabajan con estos retos y sus funciones. En este proyecto se plasman las dificultades que se presentan al proponer una solución que cubra gran cantidad de nodos y su visualización, específicamente en las limitantes computacionales de su procesamiento.

1.2.1 Análisis de genes

La biología a nivel microscópico comenzó en 1665 cuando Robert Hooke descubrió que los organismos están compuestos de compartimientos individuales llamados células. La teoría celular, ampliada por Matthias Schleiden y Theodor Schwann en los años 1830, marcó un

[1] GPU (Graphics Processing Unit) es un co-procesador que acelera las aplicaciones del CPU tomando los problemas que toman más tiempo al procesar y distribuyéndolo en los más de 100 *cores* que operan en paralelo (comparado con los 4 a 8 *cores* que tienen actualmente los CPU) [32]

punto importante en la historia de la biología. En diferentes formas, la historia de la biología se convirtió en la historia de las células. Esto marca el inicio del estudio por los genes. [2]

Actualmente se sabe que hay rasgos que son hereditarios, y lo sabemos por experiencia propia, por la herencia que nos es otorgada por nuestros antecesores (como el tipo de cabello, color de ojos, enfermedades, entre otros) o que observamos en otras especies como animales o por el tipo de frutos o formaciones en el mundo vegetal. Y sabemos también que esa herencia se da por el traspaso de genes en el momento de nuestra creación en la división celular.

Los biólogos a principios de la década de los cuarenta entendieron que los rasgos de las células eran inherentes en su información genética, que la información genética era pasada a los descendientes y que su información genética era organizada en genes que residían en cromosomas. Ellos no sabían de qué estaban hechos los cromosomas o qué hacían los genes para generar esos rasgos. Después de varios estudios llegaron a la conclusión que el rol de un gen es producir proteínas. En algún momento se pensó que un gen se encargaba de producir una proteína en específico pero después se descubrió que un gen sirve para producir una multitud de proteínas. [3]

1.2.2 Microarreglos

A finales de los noventa [4], ha existido una explosión en la tasa de adquisición de datos biomédicos. Avances en tecnologías de genéticas moleculares, tal como los microarreglos de ADN, nos permiten por primera vez obtener la vista "global" de la célula. El uso de microarreglos es una nueva tecnología con gran potencial de proveer diagnósticos médicos acertados. Ayuda a encontrar el tratamiento adecuado y la cura para muchas enfermedades y provee un retrato molecular detallado de los estados de las células a lo ancho del genoma. [5]

Los microarreglos son experimentos físicos donde en un portaobjetos se colocan moléculas de ADN en posiciones fijas, llamadas *puntos* o *características*. Se obtienen decenas de miles de puntos en un arreglo. Cada uno contiene decenas de millones de moléculas de ADN idénticas de tamaños de decenas de cientos de nucleótidos. Para los estudios de expresión genética, cada una de estas moléculas debe identificar a una sola molécula de ARNm (Ácido Ribonucleico mensajero) o *copia* en un genoma. [4]

Las muestras del portaobjetos son alteradas por un láser y escaneadas en longitudes de onda adecuadas para detectar los tintes rojos o verdes. La cantidad de fluorescencia emitida hasta la excitación corresponde a la cantidad de ácido nucleico envuelto. [6]. La muestra genera una imagen a analizar por medio de rangos de pixeles y de una manera ordenada en subarreglos, para que permita la identificación de forma directa. Muchos paquetes de software que procesan imágenes requieren parámetros adicionales para obtener el arreglo de datos que se utilizará en los modelos.

La cantidad de datos generada por este experimento hace que en la práctica sea difícil analizar y obtener resultados confiables. Se tienen dos principales características. La primera es hacer réplicas biológicas de los datos, es decir, que en las muestras se tomen datos repetidos y de esta forma obtener muestras repetidas en el mismo instante de la muestra, tomar el promedio y tener un dato más cercano a la realidad. La segunda es tener datos repetidos de forma técnica, lo cual quiere decir que al generar matrices de estados se haga un promedio de los datos en el momento del análisis.

El paso inmediato a realizar es la discretización y normalización de los datos, donde es necesario agrupar valores de la forma más sencilla posible para lograr incluirlos en algún modelo para su análisis. La mayoría de los autores toman rangos y los asocian al número de valores que quieran usar para su modelo. Por ejemplo, para el modelo booleano toman los

valores que pasen de un valor *n* y lo asocian con el valor *uno*, después toman los valores que sean menores cero igual al valor *n* y lo asocian con el valor *cero*.

1.2.2.1 Espacios de estados

Un sistema dinámico es un sistema que cambia con el tiempo, se dice que es finito si la cantidad de posibilidades que se maneja es finita. En matemática un *sistema dinámico finito (SDF)* es un par (X, f) donde el primer componente es un conjunto finito que denotamos por X, y una función f que actúa sobre el conjunto X, es decir $f: X \rightarrow X$.

Se llama **Espacio de Estados** del SDF (X, f) al grafo dirigido formado por los vértices de los vectores X y las flechas como la acción f, es decir, existe una flecha de u a v si $f(u) = v$. En el espacio de estados se ve la dinámica del sistema. [7]

¿Qué es una red reguladora genética? Los genes interactúan, cambian su intensidad, en algunos casos están apagados y en otros están muy activos dependiendo del proceso que se esté llevando a cabo. Estas relaciones entre los genes no son casuales, están determinadas por la cantidad de algunas sustancias, y de otros factores externos e internos, y esto es lo que constituye una red reguladora genética. La redes reguladoras genéticas cambian con el tiempo, ya que en algunos momentos están realizando una actividad que trae como consecuencia que la red realice otra actividad, como es la producción de proteínas, que es el final del proceso transcripción→traducción→proteína.

Supongamos que se está realizando un experimento en el tiempo, por ejemplo, agregarle una substancia a la red reguladora que altera su comportamiento habitual. Por lo que si realizamos en el tiempo *m* mediciones, obtendremos *m* vectores de *n* entradas de números reales que están relacionados en el tiempo, que es lo que se llama ***time series data***.

Las mediciones las denotaremos como sigue:

$$s_1 = (s_{11}, s_{21}, \dots, s_{n1}), \dots\dots, s_m = (s_{1m}, \dots, s_{nm}).$$

Se quiere calcular un sistema dinámico finito donde en el transcurso del tiempo se obtengan las anteriores mediciones, es decir, nos interesa buscar una función f tal que

$$f(s_{11}, s_{21}, \dots, s_{n1}) = (s_{12}, s_{22}, \dots, s_{n2}),$$

$$f(s_{12}, s_{22}, \dots, s_{n2}) = (s_{13}, s_{23}, \dots, s_{n3}),$$

$$\dots\dots\dots\dots$$

$$f(s_{1m-1}, s_{2m-1}, \dots, s_{nm-1}) = (s_{1m}, s_{2m}, \dots, s_{nm}).$$

Resolver el problema inverso de ingeniería es calcular un sistema dinámico finito (X, f) de tal forma que el *time series data* sea una trayectoria válida en el espacio de estados del SDF.

En esta tesis nos planteamos calcular el espacio de estados de un SDF conociendo una o varias funciones posibles que cumplan con el TSD, y considerando X como un conjunto con 3 o 5 estados. A cada una de estas funciones se le asigna una probabilidad de ocurrencia, y así tenemos una red reguladora probabilística.

1.2.2.2 Cadenas de Markov [8]

Es de reconocerse que en espacios de estado largos, que se encuentran comúnmente en métodos de Cadenas de Markov de Monte Carlo (MCMC) en varias aplicaciones, incluyendo el modelado de campos aleatorios de Markov para el procesamiento de imágenes, donde simulación eficiente y estimación son realizadas rutinariamente. Por lo tanto, los métodos de Monte Carlo representan una alternativa viable a métodos numéricos basados-en-matriz para

obtener las distribuciones de los espacios de estados. Informalmente se dice que este método consiste en correr la cadena de Markov por un tiempo suficientemente largo hasta que la convergencia a la distribución estacionaria se alcance, y observando la proporción del tiempo que toma el proceso en partes del espacio de estado que representa la información de interés, tales como la unión de distribución estacionaria de varios genes específicos. Un factor clave es la convergencia, la cual para gran extensión depende de una perturbación probabilística, p. En general, un p más grande resulta en una convergencia más rápida, pero haciendo a p demasiado grande no es biológicamente significativo.

Para que se pueda lograr un análisis a largo plazo de la cadena de Markov correspondiente a una PBN usando métodos de Monte Carlo, se necesita estimar la tasa de convergencia del proceso. Sólo después se puede tener la confianza necesaria de que la cadena ha alcanzado una distribución de la que se puede empezar a recolectar información de interés. Enfoques típicos para lograr la convergencia se basan en el segundo *eigenvalue* más grande de la transición de matriz A. Desafortunadamente, para un número moderado de genes, los *eigenvalues* que se obtienen de la matriz de transición pueden ser inservibles. De aquí que es ventajoso determinar el número de iteraciones necesarias hasta que una convergencia satisfactoria sea alcanzada.

1.3 Planteamiento del problema

La gran cantidad de información generada por un experimento de microarreglo son cuidadosamente guardados y almacenados en bases de datos, como los datos de la secuencia genómica, que se pueden consultar, comparar y analizar diferentes programas computacionales. Los métodos de análisis de expresión de datos se encuentran actualmente en su infancia. Junto con la secuencia genómica, la expresión sistemática de genes no es el

fin, mas bien una herramienta para crear infraestructura para investigaciones futuras. Hay un camino largo entre tener perfiles detallados de expresión de genes y el entendimiento real de los procesos celulares por debajo. Bioinformáticos y sus métodos necesitan herramientas que puedan lidiar con una gran cantidad de datos, pues estos no tendrán validez por sí solos. [4]

El problema principal que quiere resolver este proyecto es el de presentar gráficamente una cantidad de nodos mayor a mil y disponerlos de forma sencilla e intuitiva en la pantalla de un usuario con elementos que le permitan analizar adecuadamente los datos y utilizar una cantidad razonable de recursos.

1.3.1 Hipótesis

Del problema presentado anteriormente, surge la pregunta que guía este proyecto, donde se dispone resolver la problemática de más de mil nodos, utilizando recursos razonables, representarlos gráficamente y facilitando el análisis de la información.

¿Es posible poner a disposición de un usuario un modelo que presente gráficamente más de mil nodos de forma que los nodos puedan ser analizados por los usuarios involucrados en el análisis de redes reguladoras genéticas con recursos computacionales razonables?

Con el fin de responder la pregunta de hipótesis que guía este proyecto, se presentan los objetivos que deben servir como puntos guía para el desarrollo de la aplicación. La pregunta se responderá si se siguen los siguientes objetivos.

1.4 Objetivo

El objetivo principal es crear un modelo visual que genere los espacios de estados utilizando un procesamiento que permita la visualización en un tiempo razonable. El modelo a generar debe ser una descripción de los datos procesados con los parámetros que el usuario introduzca. El modelo debe presentar los resultados de forma adecuada para que el usuario pueda generar conclusiones de su análisis sobre el comportamiento que se presenta visualmente. Para cumplir este objetivo:

1. Se usará el modelo de PBN (Probabilistic Boolean Network) de Ilya Shmulevic y el Probabilistic Regulatory Network que presenta María Alicia Aviñó [7] [8] para los estudios de redes reguladoras genéticas.
2. Se generarán los espacios de estados de forma gráfica y de manera que le facilite el análisis al usuario.

1.5 Resultados esperados

Se espera tener una herramienta disponible las 24 horas del día todos los días del año en internet, con el fin de que usuarios de todo el mundo tengan a su disposición la herramienta.

Se espera que la herramienta genere adecuadamente las redes abstractamente, en un modelo lógico computacional a un tiempo razonable, que el modelo presente gráficamente y de forma clara la relación de los nodos en las redes y que provea herramientas para su mejor visualización y análisis.

Además, se busca que sea una herramienta de fácil uso donde no se necesite invertir mucho tiempo en la comprensión del funcionamiento del modelo, por lo que se espera una herramienta con pocos elementos de navegación y una distribución de elementos adecuada. También es necesario crear un tutorial que explique el funcionamiento del modelo a nivel usuario.

1.6 Justificación de la investigación

En los últimos años la biología se ha enfocado más hacia el estudio de los genes. Por ello, año con año surgen estudios y técnicas nuevas, mejoras de modelos, pruebas con algunas células, en particular E. Coli y Levadura, ya que contienen pocos genes y facilitan su estudio. Pocos estudios se han enfocado en analizar información con grandes capacidades computacionales y esto se debe a que es costoso tener una computadora con los recursos necesarios para procesar toda la información que requiere el problema.

Pocos como Beltrame, et al. (2007) [9] son los se han enfocado a hacer experimentos con *grid computing* y analisis de microarreglos. "Los métodos de análisis de datos de este tipo de expresiones se encuentran actualmente en su infancia. Incluso los enfoques más obvios, tales como análisis por cluster, y la identificación de genes expresados diferentemente, han sido usados de forma ordinaria. Así como la secuencia del genoma, los perfiles sistemáticos de la expresión de genes no son el fin en si mismo, sino más bien una herramienta para crear infraestructura para investigación futura. Hay un largo camino entre tener perfiles de expresiones genéticas detalladas y el entendimiento real de los procesos celulares a detalle. Métodos bioinformáticos y herramientas se necesitarán para ayudar a manejar las grandes cantidades de datos, pero no traerán entendimiento profundo por si mismos." [4] Es necesaria una herramienta visual que analice grandes cantidades de datos en

un tiempo razonable y que ayude al mejor entendimiento de la regulación entre los genes. Y sería de utilidad si existiese una herramienta que ayude a los investigadores a reducir el tiempo significativamente para descubrir y formular hipótesis.

El genoma humano y los estudios de las redes reguladoras genéticas tienen relativamente poco tiempo. Sin embargo, se han realizado estudios en diferentes niveles del experimento de microarreglos, como son a nivel de tecnologías para producir mejores muestras de los microarreglos en los portaobjetos, mejores formas para obtener la información de los microarreglos, diversos métodos para la discretización y normalización de los datos, mejoras a los modelos que existen para obtener mejores expresiones de genes, y formas de visualizar esa interacción en la regulación de genes.

1.7 Alcance

El proyecto cubre las expectativas de procesamiento al tratar y visualizar más de mil nodos a la vez. Esta visualización se presenta en la ventana del usuario. Sin embargo, puede realizarse el procesamiento de más nodos dependiendo de las capacidades computacionales del dispositivo que utilice el usuario. El modelo tiene herramientas básicas de manipulación de interfaz como son: "barras de desplazamiento", "herramienta de aumentar y disminuir tamaño" (*zoom*), "buscador de nodos dentro del espacio de estados creado", "impresión del modelo", "tabla de estadísticas" donde se presente el número de grafos creados, los elementos en cada grafo, el tamaño del ciclo y el valor del nodo en caso de ser un ciclo estacionario fijo.

1.8 Estructura del documento

En el presente capítulo se explican las bases generales para el desarrollo de este proyecto, plasmando los antecedentes, el planteamiento del problema –incluyendo la hipótesis generada sobre el proyecto–, los objetivos del desarrollo y los resultados que se esperan con sus límites, así como el porqué del desarrollo de este proyecto.

En el Capítulo 2 se presentan modelos existentes que sirven de base para el desarrollo del modelo actual. Los modelos se exhiben con sus funciones y limitantes, de forma tal que puedan guiar el desarrollo y apoyar la justificación del proyecto.

El Capítulo 3 trata sobre el modelo que se crea, el tipo de desarrollo que se utiliza, los requerimientos y diagramas utilizados para el desarrollo del modelo, desde el análisis y diseño del sistema, incluyendo requisitos funcionales y no funcionales, tecnologías a usar, la división del proyecto en dos subproyectos, así como los algoritmos específicos para resolver los retos presentados y el desarrollo de la interfaz gráfica.

El Capítulo 4 presenta las validaciones y pruebas que se corrieron sobre el sistema con el fin de validar su operabilidad respecto a los requerimientos funcionales, no funcionales y lo que se espera del sistema, así como sus limitantes y capacidades dependiendo de los recursos del usuario y su comparación con otros sistemas.

El Capítulo 5 presenta las conclusiones a las que se llegaron sobre el desarrollo en general del sistema, destacando el cumplimiento de los objetivos planteados y cubriendo lo que se esperaba al inicio del desarrollo, del proyecto. Se concluye también específicamente sobre temas de interés en el desarrollo como son la capacidad del sistema, tiempo y recursos que utiliza y su visión en general, extensión del proyecto a otros proyectos y lo que se logró

en el área de visualización del proyecto. Se agrega, además, una conclusión general sobre las implicaciones en el desarrollo del proyecto y su uso.

CAPÍTULO 2

Modelos de Análisis de Datos Biológicos

En este capítulo se presentan los enfoques que existen sobre el análisis y estudio de las redes reguladoras genéticas con el propósito de presentar el modelo que se tomará en cuenta para este proyecto. Además, se presentan aplicaciones de software que actúan sobre la información relacionada con el estudio de microarreglos y con el estudio específico de redes reguladoras genéticas.

2.1 Análisis de redes reguladoras genéticas

El análisis de redes reguladoras genéticas es el siguiente paso para identificar los grupos de genes que se encuentran expresados en un momento específico en el tiempo de regulación. Los objetivos del análisis de estas redes son identificar el comportamiento de los grupos de genes, así como los estados estacionarios de estos. Para lograrlo, es importante especificar tanto el tiempo de modelo que se utilizará para el estudio de las redes como las herramientas que existen con el fin de facilitar el análisis.

Acorde con Wu, Huang, Juan, & Chen (2003), [10] después de hacer la discretización de los datos es necesario presentarlos en una matriz de regulación de genes que mostrará de forma sencilla qué genes regulan a otros y cómo sus entradas genéticas afectan su expresión. La matriz se puede emplear en descubrir cómo los genes actúan en conjunto para lograr las características fenotípicas. Un usuario puede manipular cualquiera de cuatro tipos de enfoques de ingeniería inversa para el modelado de una red reguladora genetica que involucre a una matriz de regulación de genes de un conjunto masivo de datos de expresiones genéticas que contienen respuestas en tiempos medidos para varios estímulos.

Los enfoques son:

1) Redes booleanas,
2) Modelo lineal,
3) S-sistema, y
4) Redes Bayesianas Dinámicas.

Soinov, Krestyaninova, & Brazma (2003) [11] realizaron un estudio donde utilizaron datos de los genes de levadura, y su objetivo era lograr predecir el comportamiento de los genes en su regulación. Lo que hicieron fue tomar muestras y generar reglas ajustándolas cada vez que fuera necesario para cumplir con el resultado de la regulación. Por medio de inducción es que generaron las reglas. Lo que lograron fue crear un sistema de reglas donde los datos ingresados posteriormente serían analizados por estas reglas y generarían la predicción de la regulación. Utilizaron el modelo booleano y su base de reglas se basó en tres principales problemas: el primero es para genes que regulan a otros de forma simultánea; el segundo para analizar la regulación de genes a partir de muestras anteriores, y el último para ver el comportamiento de los cambios que generan los genes actuales en genes de muestras posteriores.

Después de elegir el modelo adecuado a las necesidades, se requiere procesar la información, donde algunos autores como Causton, Quackenbush, & Brazma, (2003) y Babu, (2004) [4] mencionan que el procesamiento de la información es un rubro importante que hay que tener en cuenta y ser cuidadosos, pues si cambian las variables, los parámetros de entrada, el algoritmo con el que se ejecuta, y otros, es probable que se obtengan resultados diferentes. Además, los autores mencionan que el uso de software para el análisis de redes

reguladoras genéticas se vuelve cada vez más imprescindible por sus capacidades computacionales.

2.2 Software

El software dedicado al estudio de redes reguladoras genéticas se ha ido desarrollando para solucionar problemas específicos. Algunos simplemente se enfocan en los datos de muestra y utilizan modelos matemáticos adecuados.

Existen varios tipos de software para el análisis de redes reguladoras genéticas. Algunos son en línea, algunos otros gratuitos, otros no están disponibles para cualquier usuario. Además, algunos paquetes de software se enfocan en algún proceso en especial, incluso sin mostrar un resultado de forma gráfica.

Otro software que existe actualmente, creado por Alvis Brazma, del Instituto Europeo de Bioinformática, propone sintaxis para facilitar el estudio de los datos que se ingresen a ese sistema. MAGE-TAB es un formato delimitado por espacios utilizado para describir microarreglos y potencialmente otro tipo de datos que serán utilizados por el software, MIAME 2.0 (por sus siglas en inglés *Minimum Information About a Microarray Experiment*).

Para utilizar este software, se siguen estos pasos: 1) describir las muestras biológicas, 2) describir el ensayo (como el diseño del microarreglo), 3) introducir los de los ensayos – planos y procesados, 4) generar protocolos de procesamiento de datos y materiales, 5) crear el diseño experimental donde muestra qué datos ingresaron del ensayo y qué datos produjo. Normalmente lo muestra en un grafo no-cíclico dirigido.

Estos dos tipos de software componen varios elementos que los hacen útiles para el análisis de información por parte de los biólogos pues toman la entrada y el modelo y presentan de forma gráfica para un fácil entendimiento.

Su principal limitante es la capacidad de genes a analizar, y por lo tanto la cantidad de genes regulados a mostrar, pues están pensados y construidos para ser ejecutados por una sola computadora, ya que los recursos que se necesitan no pueden ser cubiertos por los dispositivos comunes del mercado.

Grid Computing ha emergido como un campo nuevo importante, distinguido del cómputo distribuido convencional por su enfoque en compartir recursos a gran escala, aplicaciones nuevas y, en algunos casos, orientados al alto desempeño. [12]

En este contexto, existe un software similar para *Grid Computing* llamado GEMMA (por sus siglas en inglés, *Grid Environment for Microarray Management and Analysis*) propuesto por Beltrame, et al. (2007) [9], el cual utiliza Genius (*Grid Enabled web eNvironment for site Independent User job Submission*) para enviar trabajos a las diferentes computadoras que conforman la grid.

El enfoque de GEMMA se divide en dos módulos principales. El primero procesa los datos desde los genes con GeneChip™ Affymetrix para obtener los datos directamente de los microarreglos y normalizar los datos; el segundo extrae los datos normalizados para generar una matriz de expresión de genes.

Existe un paquete de software que integra la dinámica de espacios de estados. Es el llamado *Discrete Visualizer of Dynamics* (DVD), creado por el grupo de matemáticas discretas aplicadas en el Instituto de Bioinformática de Virginia. [13]

Los modelos matemáticos se utilizan frecuentemente para analizar sistemas grandes, pero los métodos computacionales actuales a veces no pueden manejar la cantidad de datos generados por investigación genética.

Dr. Laubenbacher [14] usa las complejidades celulares para representar aspectos de los sistemas biológicos y así obtener una imagen más clara de cómo funcionan las redes genéticas. Estas complejidades crean algoritmos que permitirán una mayor flexibilidad computacional y eficiencia. Aplicando teorías usadas previamente para entender patrones en modelos de tráfico a gran escala y la dispersión de patógenos, el grupo modela redes reguladoras genéticas de levadura como pruebas para su enfoque.

CAPÍTULO 3

Desarrollo del Modelo

En este capítulo se presenta el proceso de desarrollo del modelo desde que se origina la idea, cómo se concibe para transformarla lo más genuinamente posible a un sistema de información que cumpla con las expectativas finales. Además, se detallan los retos que se presentaron, así como la forma de resolver y presentar la solución.

3.1 Materiales y métodos

En este capítulo se presentan las herramientas que se utilizaron para la creación del modelo y el cumplimiento de los objetivos anteriormente planteados. Se presenta el análisis y el diseño que se hizo para desarrollar el modelo junto con sus diagramas de Lenguaje Unificado de Modelado (UML). Además, se presentan los métodos que se utilizaron con el fin de lograr que el modelo funcionara bajo los requisitos proporcionados.

El modelo se basó en el paquete de software DVD, mencionado anteriormente, por lo que los requisitos funcionales fueron definidos conforme a lo que podía hacer el programa, siempre y cuando se cumplieran los objetivos indicados en la sección 1.4.

El modelo que se presenta en este trabajo es una herramienta que permite visualizar el espacio de estados de una red reguladora genética identificando los estados estacionarios y su forma de relacionarse de una forma gráfica.

Al visualizar el espacio de estados se forma un conjunto de nodos relacionados entre sí. Cada nodo representa un conjunto de genes. La relación entre nodos se define de la siguiente forma: un nodo regula a otro, y sólo regula a exactamente uno, pero un nodo puede ser regulado por uno o más nodos. Esto forma el espacio de estados de la red y, al observar la

dinámica entre los nodos del espacio de estados, se identifica un ciclo en la red formando los estados estacionarios.

Los estados estacionarios representan el asentamiento de los nodos en la regulación, de forma que permiten generar suposiciones de qué genes están activados y el ciclo en el que participa hasta que muere la célula que producen. Por ejemplo, permiten suponer qué genes están activados cuando un pedazo de piel muere en el caso de cáncer.

3.1.1.1 Multiestados

El modelo fue pensado como un servicio a ofrecer a los biólogos o a cualquier persona que esté involucrada con el análisis de redes reguladoras genéticas. Debido a que el modelo es un desarrollo de software se utilizaron prácticas de la metodología *Programación Extrema (PE)* con un análisis y diseño más robusto, utilizando diagramas UML [2] para su mejor comprensión.

En la metodología PE el primer paso es la planeación, donde se analiza para qué está hecho el sistema y los requisitos que debe cubrir para ser funcional y que opere como se espera según los objetivos de la sección 1.4.

3.2 Análisis del modelo

A continuación se presentan los elementos de análisis que ayudaron al entendimiento del sistema, resolviendo la pregunta **¿qué hace el sistema?**. Dichos elementos se encuentran expresados en diagramas UML.

[2] UML, por sus siglas en inglés *Unified Modeling Language*, utiliza técnicas de notación gráfica

3.2.1 Requisitos funcionales

El primero de los elementos es el caso de uso. Pressman [15] expone que un caso de uso básico presenta una historia de alto nivel que describe la interacción entre el actor y el sistema.

Muchos de los requisitos se basaron en el paquete de software existente DVD. El modelo busca cumplir los objetivos de un modelo útil y usable, es decir, que haga su función principal aprovechando los recursos y que sea de un manejo sencillo, de forma que los controles sean claros y sencillos, y el manejo del modelo sea lo más intuitivo posible.

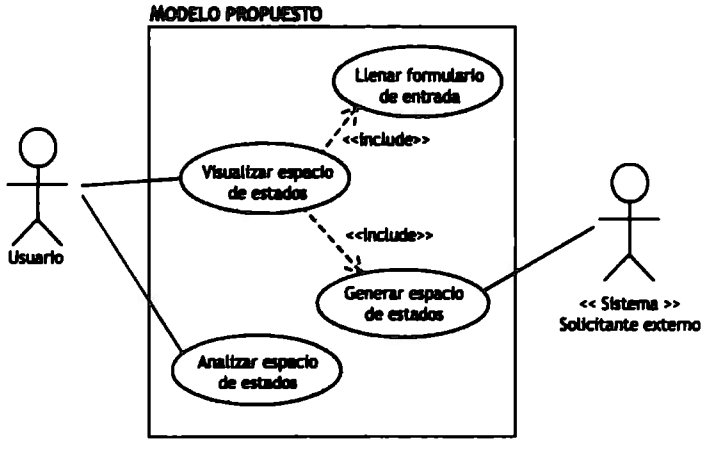


Figura 3.1 Diagrama UML de casos de uso del modelo

Con el objetivo de identificar el propósito general del modelo, se creó el diagrama de casos de uso de la Figura 3.1 donde se pueden identificar los actores Usuario y Solicitante externo. El actor Solicitante externo se explicará más adelante.

La función principal del actor Usuario en el modelo es "visualizar el espacio de estados" y "analizar el espacio de estados". Los casos de uso "Llenar formulario" y "Generar espacio de estados" son requeridos para poder visualizar el espacio de estados. A

continuación se presentan los casos de uso con una descripción de la función de cada caso de uso, así como los elementos de entrada y de salida de cada uno.

Plantillas de Cockburn

Caso de uso	Llenar formulario de entrada.
Actor primario	Usuario.
Meta en el contexto	Establecer los valores de las variables a ser enviadas.
Condiciones previas	La interfaz gráfica debe estar completamente cargada.
Activador	El Usuario hace clic sobre los campos a editar.
Escenario	El Usuario maneja el número de genes y número de estados por gen que quiere solicitar, escribe las funciones en el espacio dedicado a las funciones y al presionar el botón se enviarán las variables al caso de uso "Generar espacio de estados".
Excepciones	En el caso de que algún valor se encuentre fuera del rango se mostrará un error.

Tabla 3-1 Definición del caso de uso Llenar formulario de entrada

Caso de uso	Generar espacio de estados.
Actor primario	Usuario.
Meta en el contexto	Generar un documento en lenguaje XML[3] que contenga los siguientes elementos: o Total de nodos generados. o La raíz de cada estructura. o El ancho de cada estructura. o El tamaño de nodos de cada estructura. o El tamaño del ciclo en cada estructura. o Cada nodo de la estructura con un identificador. o Cada nodo de la estructura con un valor. o Cada nodo de la estructura con el valor de su hijo. o Cada nodo de la estructura con su nivel. o Cada nodo de la estructura con su posición en el eje *x* y su posición en el eje *y*
Condiciones previas	Las variables deben de tener valores válidos. El servidor donde se encuentre este servicio debe estar activo y recibiendo peticiones.
Activador	Se envía una petición al servicio que provee esta funcionalidad.
Escenario	Una vez recibidas las variables se valida que los

[3] XML, proviene de las siglas en inglés ***eXtensible Markup Language***, designado para transportar y almacenar datos estructurados adecuados en una red http://www.w3.org/standards/xml/core

	parámetros sean correctos. Las funciones son analizadas gramaticalmente para verificar que cumplan con la sintaxis específica. Se genera el total de nodos con la función: *total de nodos = estados ^ genes*. Se genera el total de nodos estructuralmente, es decir, los objetos en el lenguaje computacional. Se crea la estructura de datos del espacio de estados, es decir, las relaciones que forman la estructura. Se genera la posición de los nodos en un espacio bidimensional. Se crea el documento de salida.
Excepciones	En el caso de que alguna variable se encuentre fuera del rango se mostrará un error.

Tabla 3-2 Definición de caso de uso Generar espacio de estados

Caso de uso	Visualizar espacio de estados.
Actor primario	Usuario.
Meta en el contexto	Crear el espacio de estados de forma gráfica en conjunto con sus herramientas para navegar y analizar.
Condiciones previas	El documento generado por el servicio del caso de uso "Generar espacio de estados" debe estar bien formado y debe ser transferido con éxito.
Activador	Escuchador en la interfaz gráfica que registra cuándo el archivo terminó de transferir.

Escenario	Se convierte el documento recibido a objetos del lenguaje computacional. Se crean las relaciones entre los objetos creados para después colocarlos gráficamente en su posición *x* y *y*. El espacio de estados debe verse todo junto y no en partes
Excepciones	Ninguna.

Tabla 3-3 Definición de caso de uso Visualizar espacio de estados

Caso de uso	Analizar espacio de estados.
Actor primario	Usuario.
Meta en el contexto	Permitir la navegación útil y sencilla donde el usuario pueda analizar datos fácilmente, así como buscar nodos dentro del espacio de estados.
Condiciones previas	El espacio de estados debe estar generado, así como sus herramientas de navegación y el resumen del espacio de estados.
Activador	Un mensaje de alerta que indique el término de la generación del espacio de estados.
Escenario	Con el espacio de estados creado, se verifican las estadísticas creadas, los ciclos que lo componen, así como el número de nodos que la conforman. Sobre estas estructuras es que se puede hacer una búsqueda de un nodo en específico.
Excepciones	Ninguna.

Tabla 3-4 Definición de caso de uso Analizar espacio de estados

En las tablas 3-1, 3-2, 3-3 y 3-4 se presentaron las definiciones de los casos de uso que hacen que el modelo funcione como se espera. Además de estos requerimientos, se presentan los requerimientos no funcionales que ayudará a que el modelo opere de forma adecuada y ofreciendo el servicio que se espera.

3.2.2 Requisitos No funcionales

Bajo nivel de uso de procesador y de uso de memoria. El modelo podría generar un uso intenso del procesador y la memoria en sus procesos de generar las estructuras y recorrerlos.

Navegación intuitiva y sencilla. Es necesario que la interfaz sea simple para su uso e intuitiva para mejorar las funciones de análisis de datos.

Visualmente atractivo. Ayudará a que el usuario se sienta cómodo con el uso del modelo.

Ligero. Debe ser de componentes robustos pero que el tiempo de carga de la interfaz sea de segundos.

Multiplataforma. Que el modelo pueda ser utilizado por las diferentes plataformas existentes (Windows, Mac, Linux y otros).

Disponibilidad. El modelo debe ser presentado en un esquema web, disponible todo el tiempo.

Accesibilidad. El modelo debe ser de acceso público, de forma que cualquier persona que quiera utilizarlo lo haga sin restricciones.

Seguridad. El modelo debe proteger la seguridad del equipo del usuario y del servidor en el que se encuentre publicado.

Portabilidad. El modelo debe ser accesible en el mayor tipo de equipos posibles (computadoras de escritorio, portátiles, móviles, entre otros).

Precio. El uso y almacenamiento del modelo no deben de presentar costo alguno para los usuarios ni para los creadores.

3.2.3 Flujo del modelo

Se han presentado los requisitos del modelo para que funcione como se espera. Es necesario también mostrar el flujo en el uso del modelo desde la perspectiva del usuario.

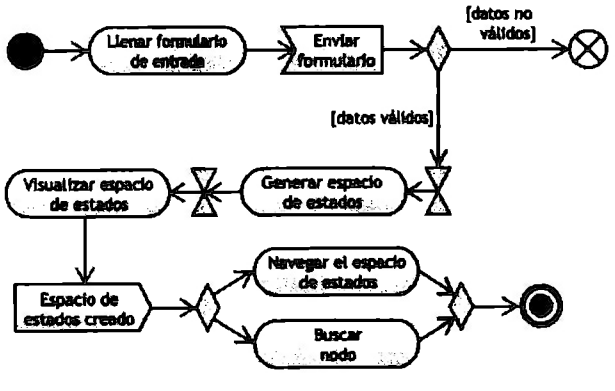


Figura 3.2 Diagrama UML de actividad del modelo

La Figura 3-2 muestra el diagrama de actividad iniciando con el proceso de Llenar el formulario de entrada y esperando a que se envíe el formulario presionando el botón de Generar espacio de estados.

Los datos se validan y si no son correctos termina el proceso sin enviar los datos. Si son correctos se envían y toma tiempo para que el espacio de estados sea generado. La razón del tiempo de espera es por la complejidad computacional de dos problemas principales:

1. Representar la estructura de datos computacional.
2. Recorrer la estructura y formar relaciones o ejecutar funciones dada su representación computacional. En este caso, generar el posicionamiento entra dentro de las complejidades.

Una vez generado, toma tiempo en ser visualizado, pero cuando termina muestra un mensaje de Espacio de estados creado. El tiempo de espera en este punto es por otro problema computacional: la representación gráfica del espacio de estados se puede elegir entre crear una imagen de mapa de bits, un documento pdf con hojas o un documento en vectores entre otros.

Con el espacio de estados visible, se puede o navegar en el espacio de estados o buscar un nodo. Cualquiera de estas dos actividades son actividades terminales en el flujo para la presentación de los grafos.

3.2.4 Estructura del modelo

En la etapa de análisis, el diagrama de clases es usado para representar la estructura del modelo en forma general. En el análisis del desarrollo del modelo se llegó a la conclusión de formar cuatro entidades. Como se puede apreciar en la Figura 3.3 existe una entidad Modelo

que generará los nodos y generará visualmente el espacio de estados. Una entidad que servirá de analizador gramatical para las funciones. Una tercera entidad representará el nodo, sus padres, su valor y su posición en un espacio bidimensional. Una última entidad representa la línea que une a los nodos gráficamente.

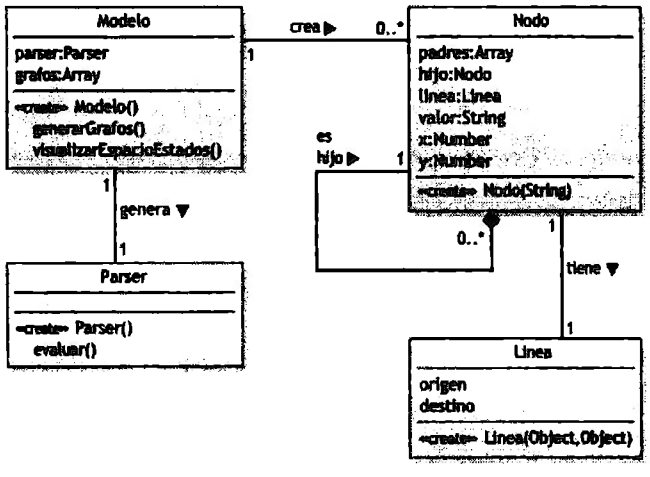


Figura 3.3 Diagrama UML de clases

La forma de la estructura se ve reflejada en la relación "tiene padre" y en la relación "tiene hijo" de la entidad Nodo. Un Nodo tiene exactamente un hijo pero un Nodo puede ser hijo de 0 o muchos padres, lo que ocasiona que el último hijo tenga como hijo a sí mismo o a alguno de sus antepasados, formando así un ciclo en la estructura. La estructura cambiará en la etapa de diseño para mostrar los detalles de implementación que sufrirán las entidades.

3.3 Diseño del modelo

En esta etapa se pretende formar un marco de trabajo para el desarrollo del modelo, considerando los requerimientos funcionales y no funcionales, de modo que se puedan apreciar los detalles para la construcción del modelo.

Configuraciones de nodos y conexiones ocurren en una gran diversidad de aplicaciones. Ellas pueden representar redes físicas, tales como circuitos eléctricos, carreteras o moléculas orgánicas. Ellas también son usadas en representar interacciones menos tangibles como podrían ocurrir en ecosistemas, relaciones sociológicas, bases de datos o en el flujo de control en un programa computacional. [16]

Uno de los objetivos del modelo presentados en la sección 1.4 es el de crear visualmente las relaciones entre nodos. Estas relaciones son unidireccionales. Un grafo dirigido o digrafo $G=\{V,E\}$ consiste de un número finito, no vacío de vértices V y un conjunto de aristas E. Cada arista es un par ordenado (v,w) de vértices. [16]

3.3.1 Complejidades computacionales

La estructura computacional se forma con base en el número de genes y el número de estados por gen por medio de la fórmula:

T = Total de nodos del espacio de estados

n = Número de genes

s = Número de estados por gen

$$T=s^n$$

La primer complejidad que se identifica es el número de nodos a procesar. El proceso utiliza recursiones en sus algoritmos, lo que hizo que se pensara en un enfoque dividir-y-conquistar.

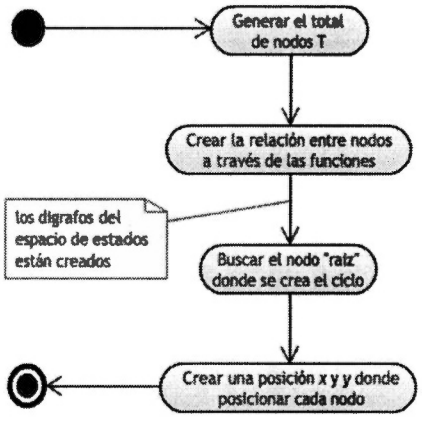


Figura 3.4 Diagrama UML de actividad para generar espacio de estados

Muchos algoritmos útiles son recursivos en estructura: para resolver un problema dado, se llaman a ellos mismos recursivamente una o más veces para tratar con los subproblemas cercanamente relacionados. Estos algoritmos siguen típicamente un enfoque dividir-y-conquistar: rompen el problema en varios subproblemas que son iguales al problema original pero más pequeños en tamaño, resuelven los subproblemas recursivamente, y luego combinan estas soluciones para crear una solución al problema original. [17]

El enfoque dividir-y-conquistar sugirió utilizar *grid computing* para atacar el problema de generar el espacio de estados, dividiendo el espacio de estados en sub-digrafos, y a partir del paso 3 en la Figura 3.4 dividirlo en el total de dispositivos conectados al grid y esperar el procesamiento.

Usar *grid computing* en este caso, y limitado para la forma en que se procesa la red reguladora genética, retrasaría el procesamiento de grafos, debido a que para cada grafo hay que tomar en cuenta todos los nodos para poder crear las relaciones, es decir, por cada computadora en el *grid*, y tendría que procesar lo mismo que procesa una sola computadora.

Una de las razones de no usar *grid computing* según el proceso actual, es que no hay un punto en el proceso por el cual sea conveniente dividir el procesamiento, sin que afecte, además, el tiempo de comunicación entre el *grid* y el controlador central. *Grid computing* requiere que el software se pueda dividir y la reorganización de las diferentes piezas [18]. en este caso, el análisis de nodos no podría dividirse debido a que el procesamiento de redes requiere analizar todos los nodos.

Una de las razones para usar *grid computing* sería el hecho de generar las imágenes de los subgrafos y pasarlas al controlador central del *grid* y unirlas, aprovechando de esta forma el procesamiento en paralelo para crear cada imagen. Sin embargo, para poder crear este modelo, hay varios puntos que toman tiempo: el dividir el documento XML que produce Gefundus en varios documentos, el tiempo para pasar el documento a todas las computadoras al grid, el generar la imagen de cada uno, la descarga de cada imagen desde el servidor y el unirlas en una sola imagen.

Mostrar las imágenes en un solo espacio es otro de los problemas que se presentan en el proyecto que grid computing no puede resolver, debido a que cuando todas las imágenes estén completas y descargadas, se debe encontrar una forma de unirlas y desplegarlas en un espacio considerable.

Debido a que se tenían complejidades en el proyecto con más importancia para poder mostrar el resultado final, que son los espacios de estados, la ventaja de usar grid computing únicamente para el procesamiento de imágenes se dejó para futura investigación.

Generar los elementos gráficamente supone un mayor problema para que el modelo funcione como se espera debido a que, una vez que se tienen las posiciones x y y de cada nodo, hay que generar una imagen ya sea en mapa de bits o en vectores que represente la

cantidad de nodos procesados de una forma legible y clara para el usuario. El proceso de crear la imagen global del espacio de estados es, entonces, el proceso que requiere de más recursos computacionales de un solo procesador. Más adelante se muestran las tecnologías utilizadas para representar el espacio de estados.

En la Figura 3.2 se muestran los dos procesos que ocasionan la mayor complejidad computacional, Generar espacio de estados y Visualizar espacio de estados. Se decidió dividirlos en dos proyectos que se comunican por las siguientes razones:

1. Un proyecto se dedicaría a Generar el espacio de estados y el otro a representar gráficamente el modelo.
2. Se ofrecería la posibilidad de extender el proyecto que genera el espacio de estados a correrlo sobre alguna de las tecnologías que permiten dividir recursos computacionales como *grid computing* o *cloud computing*[4].
3. Se ofrecería un servicio a otras aplicaciones que decidan mostrar gráficamente el espacio de estados.
4. Facilidad en el entendimiento y mantenimiento del código generado.

El proyecto que genera el espacio de estados fue llamado Gefundus y el proyecto que visualiza el espacio de estados y que sirve también de interfaz gráfica para la entrada de variables fue llamado Genodus.

[4] *Cloud Computing* es un beneficio para empresas que quieren externalizar completamente su infraestructura de centro de datos. [18]

3.3.2 Proyecto Gefundus

El proyecto Gefundus surge con el propósito de generar el espacio de estados y ofrecer el servicio a cualquier aplicación que lo solicite. Este proyecto cubre el caso de uso Generar espacio de estados de la Figura 3.1 y cumple con los requisitos de la Tabla 3-2.

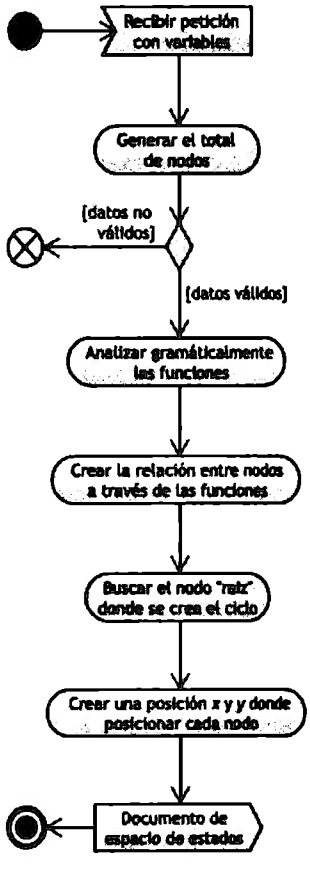


Figura 3.5 Diagrama UML de actividad de Gefundus

El diagrama de la Figura 3.5 muestra que el flujo del sistema inicia al recibir la petición con las variables, que después ayudará a generar el total de nodos. Si los datos no son válidos envía un error; de lo contrario, continúa con el flujo analizando gramaticalmente las funciones, creando la relación entre nodos y, después, generando la posición vertical y horizontal del nodo.

Es importante también mostrar el diagrama de máquina de estados de la Figura 3.6 Diagrama UML de máquina de estados de Gefundus que ayuda a comprender las fases por las que pasa este proyecto. Primeramente se queda esperando una petición. Al recibirla, cambia al estado donde valida los datos. Una vez validados, inicia el proceso de generación del espacio de estados, que se termina con el documento creado.

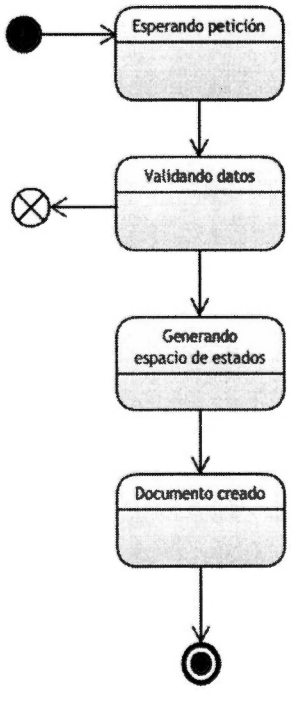


Figura 3.6 Diagrama UML de máquina de estados de Gefundus

La estructura de Gefundus se basa en la Figura 3.3 agregando detalles importantes de su implementación y quitando los que pertenecen al proyecto Genodus. En la Figura 3.7 se muestra el Diagrama de clases de Gefundus y sus relaciones. Gefundus está formado por seis clases. Las clases principales son: GenodusServlet, que recibe la petición; Genodus, que controla y genera el espacio de estados, y Nodo, que crea los grafos en sí mismo.

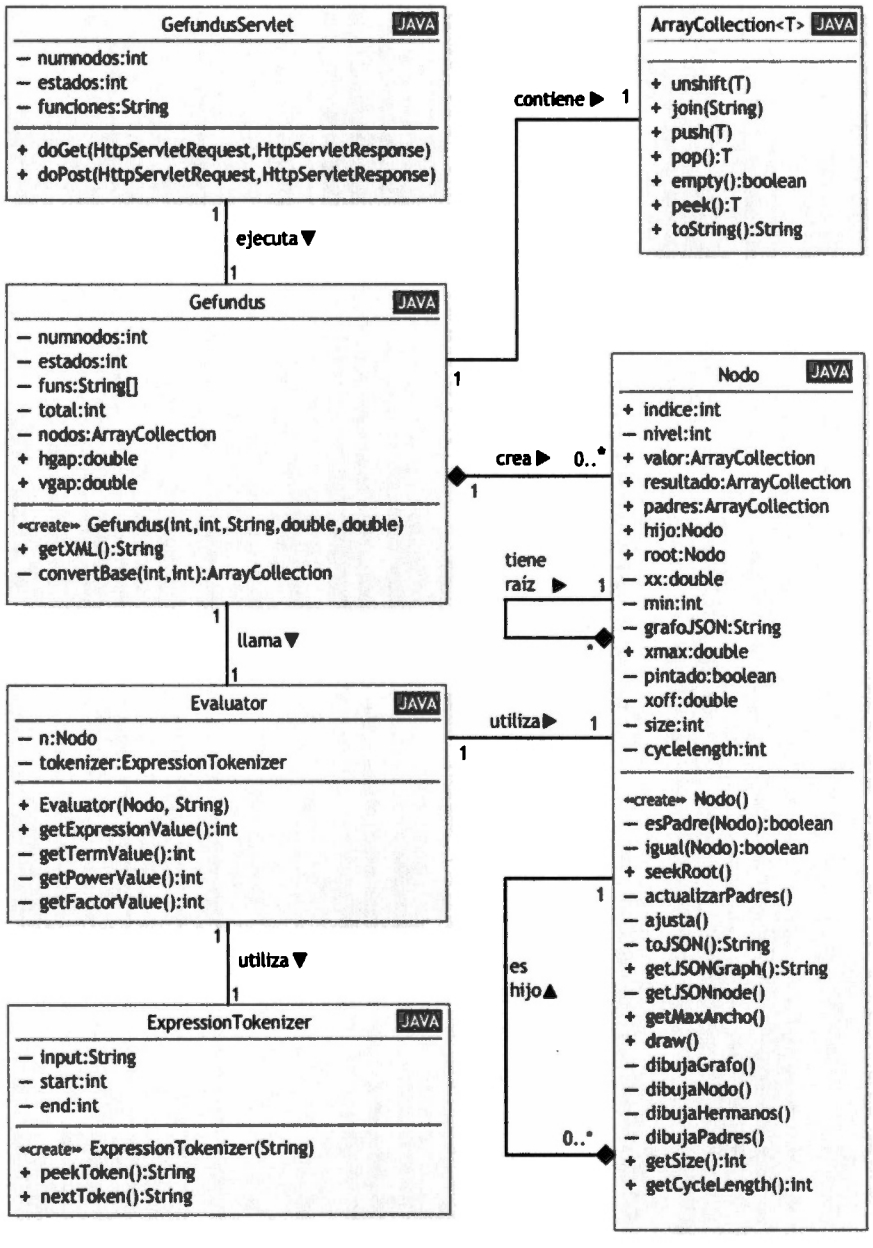


Figura 3.7 Diagrama UML de clases de Gefundus

Como se mencionó anteriormente, Gefundus se enfoca en la generación de espacio de estados únicamente, por lo que puede ofrecer sus servicios a aplicaciones gráficas que decidan mostrar de forma diferente el espacio de estados. El proyecto Genodus fue hecho para complementarse con el proyecto Gefundus al encargarse de la interfaz gráfica y la relación con el usuario.

3.3.3 Proyecto Genodus

El proyecto Genodus surge con el propósito de ser la interfaz que comunique al usuario con el sistema. Este proyecto cubre los casos de uso Llenar formulario de entrada, Visualizar espacio de estados y Analizar espacio de estados de la Figura 3.1.

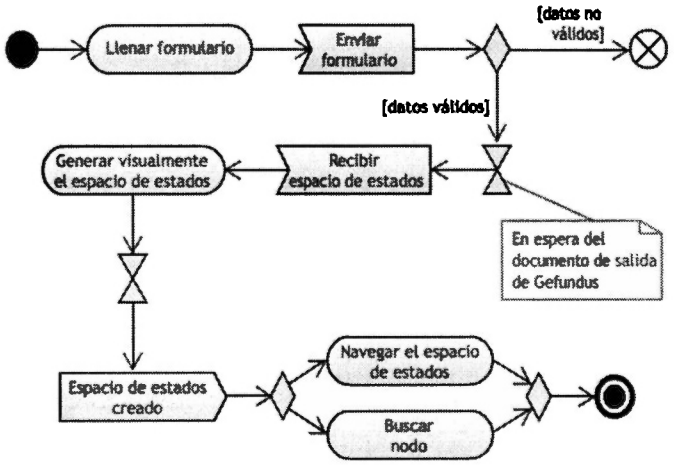


Figura 3.8 Diagrama UML de actividad de Genodus

El diagrama de la Figura 3.8 muestra el flujo del uso de Genodus. Primero se llena el formulario, se envía y se validan los datos. Si hay error se termina el flujo; de lo contrario, continúa y espera a que Gefundus envíe los datos del espacio de estados. Cuando se reciben, se genera visualmente el espacio de estados y aquí hay que esperar a éste se genere. Una vez creado, aparecerá un mensaje de terminación y se podrá navegar en el espacio de estados o buscar un nodo. Después termina su flujo.

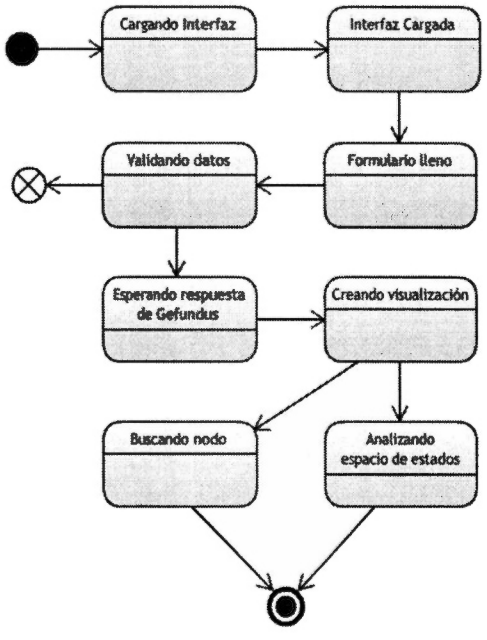


Figura 3.9 Diagrama UML de máquina de estados de Genodus

Genodus, al ser la interfaz con el usuario, resiente las cargas de uso de procesador y de memoria, por lo que el diagrama de la Figura 3.9 identifica los estados por los que atraviesa este proyecto. Primero tiene que cargarse la interfaz, con todos sus elementos normales. Una vez cargada, el siguiente estado es Formulario lleno. Al pasar por el estado Validando datos hay dos posibilidades: que se provoque un error o que se envíen y esperar la respuesta de Genodus. Este estado tomará alo de tiempo mientras se genera el espacio de estados. Ya que termina de crearse, pasa al estado de Creando visualización, en el que una vez más tomará algo de tiempo en generarse la visualización de nodos. Cuando termine de crearse la interfaz puede encontrarse en el estado de Buscando nodo o en el estado Analizando espacio de estados, donde terminan los estados.

Genodus utiliza varios elementos para mostrar gráficamente el espacio de estados, por lo que el diagrama en la Figura 3.10 representa una parte del diagrama de clases de la Figura 3.3 pero no utiliza la clase encargada de analizar gramaticalmente las funciones. Sin embargo, agrega otras y lo presenta en forma detallada.

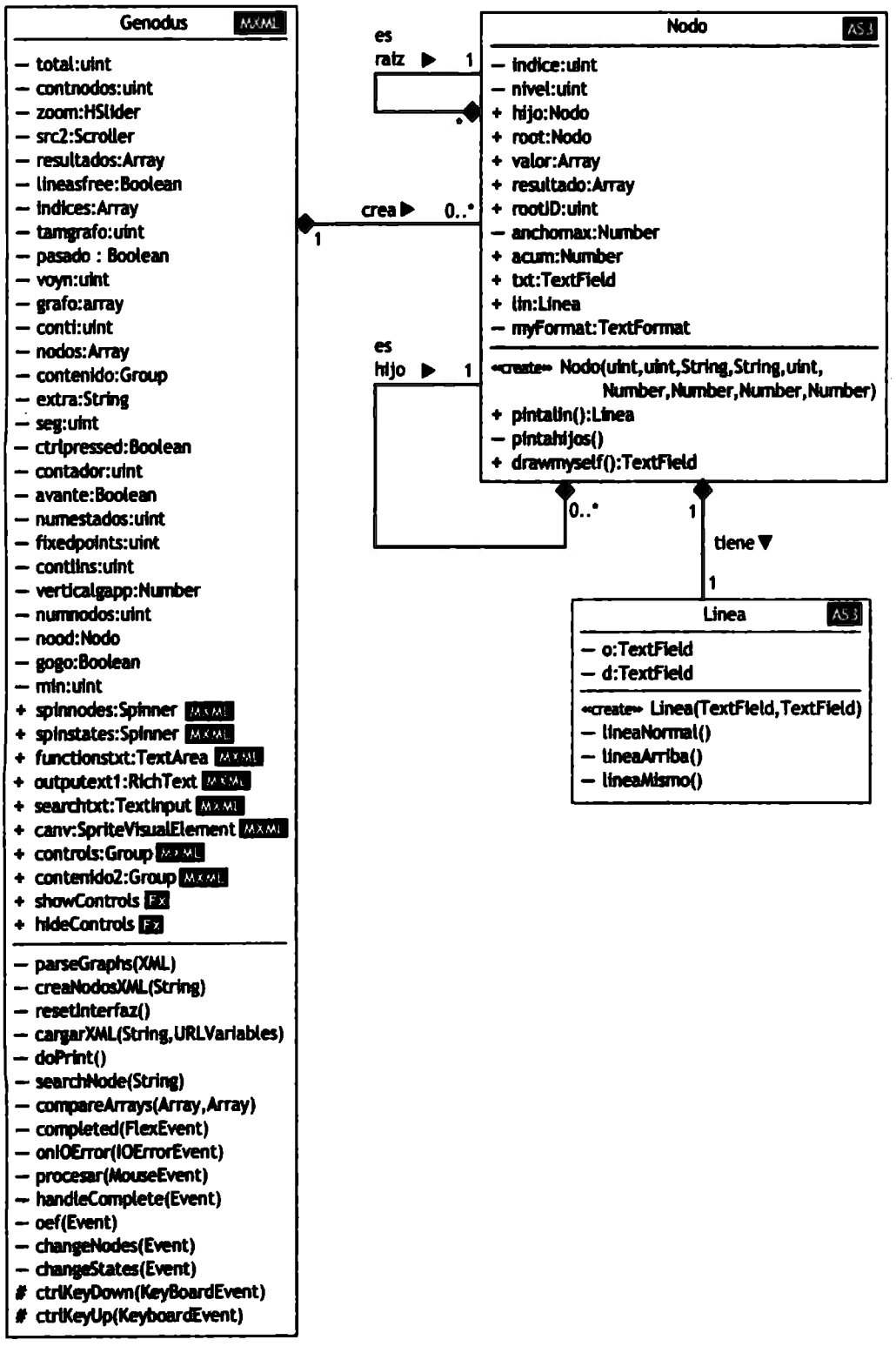


Figura 3.10 Diagrama UML de clases de Genodus

El diagrama de clases de Genodus sólo tiene tres entidades. La principal, marcada con MXML en la esquina superior derecha, es la que contiene lo gráfico de la interfaz, imágenes, cajas de texto, botones y controles MXML, además de las variables con las que interactúa, así como un efecto de movimiento gráfico manejado como variable por el lenguaje utilizado. La entidad Nodo es la que en realidad se muestra gráficamente por cada nodo donde cada uno de ellos puede tener un solo hijo, pero un hijo puede tener cero o muchos padres. Además, la entidad Linea pertenece a la entidad Nodo una vez, es decir, un Nodo sólo puede tener una línea, la que va a su hijo.

Con el fin de distinguir los diferentes lenguajes de programación utilizados se marcaron las entidades en la Figura 3.7 y Figura 3.10 con una etiqueta superior derecha con la palabra JAVA, MXML y AS3 (*Actionscript* 3.0), que representa los lenguajes de programación con los que fueron desarrolladas dichas entidades. A continuación se presentan las tecnologías utilizadas y la justificación del uso de éstas.

3.3.4 Tecnologías utilizadas

El uso de las tecnologías para el desarrollo del modelo se alinea con los requisitos funcionales y no funcionales presentados anteriormente. A partir de la decisión de dividir el modelo en dos proyectos, se piensa en la tecnología involucrada para lograr que cada proyecto logre sus objetivos y se obtenga un resultado en conjunto exitoso.

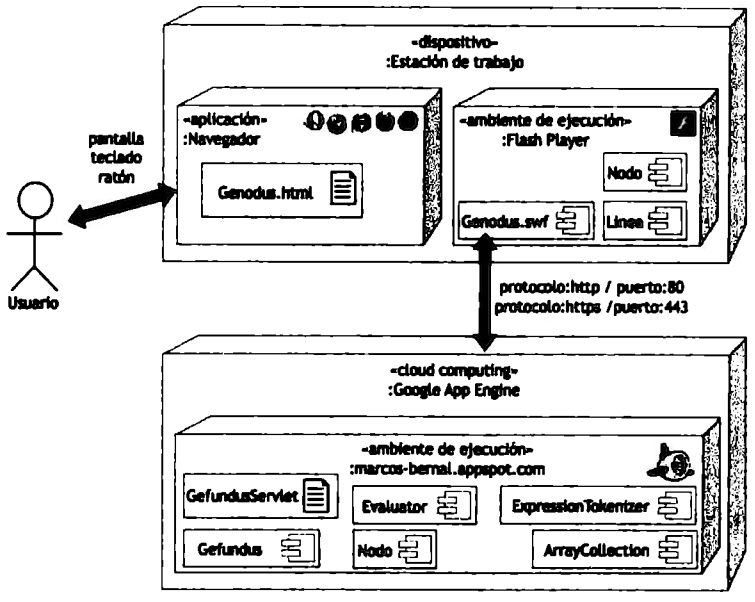


Figura 3.11 Diagrama UML de despliegue del modelo

El diagrama en la Figura 3.11 permite conocer la distribución de los elementos que permiten que el modelo funcione. Se puede ver el actor en comunicación con la página que carga el navegador del equipo del cliente desde el servidor de *Google App Engine*[5] por medio de la pantalla, el teclado y el ratón. Además, se observa el ambiente de ejecución Flash Player que ejecuta el archivo Genodus.swf, objetos de la clase Nodo y objetos de la clase Linea. Genodus entonces se está desplegado del lado del cliente y se comunica por el protocolo http en el puerto 80 o con el protocolo https en el puerto 443 para enviar y recibir peticiones hacia Gefundus.

El proyecto Gefundus está desplegado sobre la tecnología *cloud computing* de *Google App Engine*. En este ambiente de ejecución, se almacenan y ejecutan todos los componentes de Gefundus, empezando por el archivo dinámico GefundusServlet, que recibe peticiones y envía el resultado de lo que ocurre en Gefundus.

[5] *Google App Engine* es un servicio que permite ejecutar aplicaciones sobre la infraestructura de Google, [31]

Los proyectos fueron desarrollados con la intención de aprovechar al máximo los recursos disponibles, entendiendo por estos la memoria que ocupan los elementos generados, el tiempo de ejecución y el procesamiento del servidor, además del procesamiento del cliente.

El desarrollo de Genodus fue desarrollado con *Adobe® Flex®* 4.0 por las siguientes razones:

- Experiencia del autor para desarrollar con esta herramienta y con el lenguaje *Actionscript* 3.0.
- Se ejecuta en el dispositivo del cliente.
- Se ejecuta sobre *Flash Player,* lo que permite que sea multiplataforma.
- Facilidad y rapidez para crear la interfaz gráfica, junto con el objetivo de ser atractiva visualmente.
- Sencillez de desarrollo debido a las funcionalidades de componentes spark de *Flex.*
- Uso de objetos vectoriales, lo que permite que los nodos sean creados sin utilizar tantos recursos como lo hace un mapa de bits.
- Permite la fácil integración de Aplicaciones Ricas en Internet (RIA, por sus siglas en inglés, *Rich Internet Application*) a través de llamadas asíncronas a un servidor.
- *Flash Player* está entre las plataformas de software más populares del mundo, usado por más de 3 millones de profesionales alcanzando un 99 por ciento de computadoras de escritorio, así como un amplio rango de dispositivos en mercados maduros, entendiendo por estos Estados Unidos, Canadá, Reino Unido, Francia, Alemania, Japón, Australia y Nueva Zelanda.

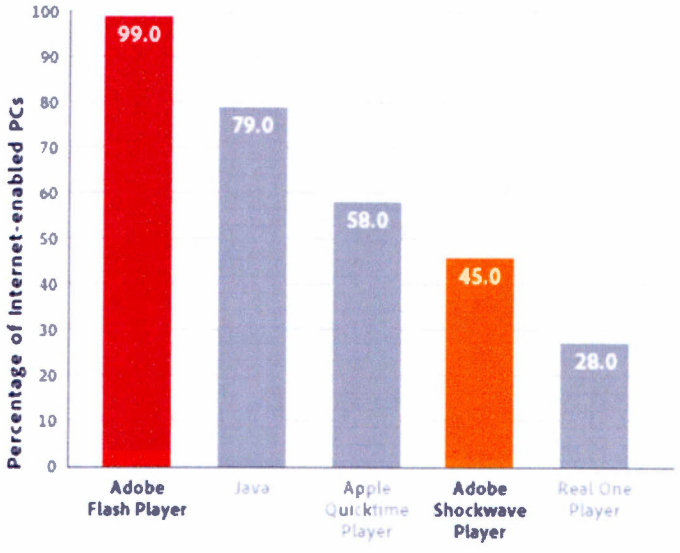


Figura 3.12 Uso de plataformas de software en internet de mercados maduros.[6]

El desarrollo de Gefundus fue desarrollado en JAVA 5.0 por las siguientes razones:

- Experiencia del autor para desarrollar con este lenguaje.
- Soporte del lenguaje para páginas dinámicas del lado del servidor.
- Lenguaje orientado a objetos.
- Velocidad de procesamiento (entre los lenguajes conocidos por el autor).
- Capacidad de ejecutarse en cualquier dispositivo con un servidor que soporte JAVA.
- Es uno de los lenguajes más populares actualmente (Raul Chong, Noviembre 16, 2010).

El despliegue de Gefundus es sobre *Google App Engine* por las siguientes razones: [7]

- Ofrece el 99.9 por ciento de servicio continuo, lo que se traduce en los requisitos no funcionales, disponibilidad.
- Escalabilidad automática. Quiere decir que si por alguna razón necesita más recursos, los toma de otras computadoras con el sistema GFS.

[6] Tomada de http://www.adobe.com/products/player_census/flashplayer/
[7] Tomada de http://code.google.com/appengine/

- Se toman en cuenta los servidores confiables de Google y la experiencia de 10 años en seguridad, desempeño y procesamiento de gran escala.
- Soporta JAVA.
- El servicio es gratis con 500Mb de espacio y 5M de visitas mensuales. En caso de requerir más recursos, la empresa ofrece servicios con costos a la medida.

Se ofrece una Arquitectura Orientada a Servicios (SOA por sus siglas en inglés, *Service Oriented Architecture*) que provee el servicio de generar el espacio de estados en un documento en formato XML o JSON. El servicio se ofrece consultando la siguiente dirección de internet:

http://marcos-bernal.appspot.com/gefundus

En necesario que este servicio reciba tres parámetros en los métodos POST ó GET compatibles con HTTP/1.1, que son los siguientes:

- *nodes*: entero mayor a cero.
- *states*: entero mayor a cero.
- *functions*: conjunto de funciones.

Si no se reciben estos parámetros, o no son válidos, el servicio mostrará un mensaje de error como respuesta a los parámetros incorrectos.

El uso de las tecnologías mencionadas cubre gran parte de los requisitos no funcionales y facilitan la interacción entre las estructuras de datos y sus componentes para lograr que el modelo se presente con la información esperada.

3.4 Desarrollo del modelo

El análisis de la sección anterior sirvió de guía para desarrollar el modelo, donde los elementos de programación se tomaron en cuenta cuidadosamente para cumplir con los requisitos funcionales y no funcionales de la sección 3.2.2 y, principalmente, para cubrir los objetivos de la sección 1.4.

Como se mencionó anteriormente, el modelo se dividió en dos proyectos: 1) Gefundus, dedicado al procesamiento del espacio de estados de forma lógica, donde el resultado es un documento en texto, y 2) Genodus, dedicado a la comunicación con el usuario, que crea peticiones y se representa el espacio de estados de forma gráfica. Se desarrolló primero Gefundus, con el objetivo de tener funcionando el servicio y después adaptar el ambiente gráfico de Genodus conforme al documento generado por Gefundus.

3.4.1 Desarrollo de Gefundus.

Debido a que el propósito principal de Gefundus es crear el espacio de estados en forma lógica, es en este proyecto donde se encuentran los algoritmos más importantes del modelo. A continuación se presentan los algoritmos que componen la sección de Gefundus en el orden de activación conforme al diagrama de actividad de la Figura 3.5.

3.4.1.1 Algoritmo de entrada y de salida del servicio Gefundus

Este algoritmo pertenece a la clase GefundusServlet de la Figura 3.7 y se encarga principalmente de recibir los parámetros por el método GET o POST, instanciar la clase principal de Gefundus e imprimir el documento que será recibido por el servicio que convierta los datos gráficamente, en este caso, Genodus. Se presenta el pseudocódigo del algoritmo.

```
1 Pedir parámetros
2 Var g = new Gefundus (parámetros)
3 imprimirGefundus()
```

El código de este algoritmo es relativamente sencillo, al encargarse de pedir los parámetros en la línea 1, crear el modelo en Gefundus en la línea 2 e imprimir el modelo en la línea 3.

3.4.1.2 Algoritmo de creación del espacio de estados Gefundus

El algoritmo en la clase Gefundus es el encargado de crear el espacio de estados y de prepararlo para impresión. A continuación se presenta el algoritmo en pseudocódigo.

```
1 Gefundus(estados, numnodos, strfuns){
2   Parse strfuns
3   var total= estados^numnodos
4   var nodos=new ArrayCollection()
5 }
```

En esta sección del código se convierten las funciones de texto a código ejecutable, es decir, de tener una función que se recibe como f1=x1^2 a tenerla preparada para sustituir valores y evaluar. Además, se crea el total de nodos a representar en el espacio de estados. Este número es el número de estados elevado al número de nodos iniciales, generando así todas las combinaciones de números posibles. Se crea además el espacio que almacenará los nodos procesados.

```
1 //Crear nodos y sus hijos
2 for(int i=0;i<total;i++){
3   Nodo n=new Nodo();
4   n.indice=i;
5   n.valor=convertBase(i,estados);
6   n.resultado=evaluarFunciones();
7   nodos.add(n)
8 }
```

En este segmento del código se convierte de un índice consecutivo a su valor correspondiente en la base que se maneja actualmente. Por ejemplo, si es base 3, los valores 0, 1, 2, 3, 4, 5, 6, 7 y 8 se convertirán a 000, 001, 002, 010, 011, 012, 020, 021, 022 y así sucesivamente hasta el total de nodos del espacio de estados; éste es el valor correspondiente al nodo y es por eso que se asigna en la línea 9. El resultado del nodo es la salida del proceso de evaluar las funciones que introduce el usuario para cada "gen" del nodo, es decir, si se tiene el nodo con los valores 022 y las funciones f1=x3^2, f2=x1+x2 y f3=x2^3+x3 el resultado será 120. Por último, en este segmento de código se agrega el nodo procesado a la colección de nodos en la línea 11.

3.5 Algoritmo de recorrido

Una vez creados los nodos con su resultado, lo siguiente es crear la relación abstracta en los datos.

```
1  //Generar grafo
2  for(int i=0;i<total;i++){
3    n1=nodos.get(i);
4    for(int j=0;j<total;j++){
5         Nodo n2=(Nodo)nodos.get(j);
6         if(n2.valor.equals(n1.resultado)){
7              n2.padres.push(n1);
8              n1.hijo=n2;
9         }
10   }
11 }
```

En el segmento de código anterior, se crean los grafos; se recorre la colección de nodos y se va revisando nodo por nodo (n1 en la línea 3) sobre todos los demás nodos (ciclo en la línea 4), y si el valor del nodo (n1) es igual al resultado de cualquiera de los demás o de sí mismo (n2), quiere decir que n1 es hijo de n2 (línea 8) y que n2 es padre de n1 (línea 7).

Ya que se ha creado la estructura de datos, lo siguiente es encontrar un punto de referencia en el grafo para crear gráficamente el árbol. A este punto de referencia se le llamó raíz.

```
1 //Buscar "raíz" y generar niveles
2 for(int i=0;i<total;i++){
3   Nodo nu;
4   nu=nodos.get(i);
5   nu.seekRoot();
6 }
```

El segmento de código anterior ordena los grafos buscando la raíz y asignando a cada nodo una distancia a partir de este punto de referencia, a la que se le llamó nivel. La implementación de este método se ve en la implementación de los algoritmos de la clase Nodo; por ahora es suficiente con saber que se toma un punto de referencia y se ordena el grafo para que se pueda pintar gráficamente.

El algoritmo de seekRoot() tiene dos propósitos: buscar un nodo como raíz artificial y asignar niveles a los nodos. En el caso de los grafos que puedan tener una raíz en uno o más nodos que están al mismo nivel, se tomo como raíz el primer nodo que se encuentre en el ciclo, y ajusta los niveles dentro de cada grafo de tal forma que todos tengan un nivel de cero o más.

Los grafos siempre tendrán un ciclo de uno o más nodos, y el nodo raíz siempre estará dentro del ciclo. El algoritmo que organiza el grafo lógicamente tiene la siguiente secuencia:

Buscar raíz

a. Para todos los padres, si el nodo actual es cíclico y además no tiene raíz,
    i. Asignarlo como raíz.

ii. Actualizar a todos los nodos del grafo.

iii. Ajustar los niveles a números naturales.

b. Si no tiene padres, buscar a través del hijo

Actualizar padres

1. Para todos los padres:

   a. Restar un nivel desde la raíz.

   b. Indicar quién es la raíz del grafo.

   c. Indicar al nodo raíz el valor mínimo de niveles.

   d. Actualizar padres para cada padre.

Ajustar niveles:

2. Para todos los padres:

   a. Aumentar en todos los niveles el valor mínimo en su valor absoluto.

   b. Ajustar Niveles de cada padre.

Después de tener ordenados los nodos en cada grafo, lo siguiente es pintarlo.

```
1 //Posicionar en una coordenada 'x' y 'y'
2 for(int i=0;i<total;i++){
3   n1=nodos.get(i);
4   n1.draw();
5 }
```

Como se mencionó anteriormente, Genodus es el encargado de representar gráficamente el espacio de estados. Lo que hace el segmento de código anterior es asignarle un valor *x* y *y* dentro del grafo con la intención de pasar esta posición al encargado de representar gráficamente el espacio de estados (en este caso Genodus) y sólo se dedique a crear el gráfico y posicionarlo en las coordenadas dadas sin tener que hacer más cálculos.

El código crea las posiciones de los nodos dentro de cada grafo. Después de crear las posiciones de cada nodo, es necesario obtener el ancho del grafo "pintado".

```
1 //Obtener el ancho máximo
2 for(int i=0;i<total;i++){
3   n1=nodos.get(i);
4   if(n1==n1.root){
5        n1.root.getMaxAncho();
6   }
7 }
```

El segmento de código anterior se encarga de obtener el ancho de cada grafo recorriendo cada nodo raíz y ejecutando la función getMaxAncho, que guardará nivel por nivel el nodo que esté en la mayor posición horizontalmente.

Después de esto se encuentra el código de imprimir que genera el documento a ser consumido por otro usuario o sistema.

```
1 imprimirGefundus(){
2    var documento=""
3    for(int i=0;i<total;i++){
4       if(nodo es raíz){
5          imprimeGrafo()
6       }
7    }
8 }
```

Es importante presentar además la función convertBase() que convierte directamente un número dado en la base recibida en los parámetros.

```
1  convertBase(numero, base){
2
3     var nuevo=new ArrayCollection()
4     var tmp=numero
5     mientras (tmp > base){
6        nuevo.unshift( tmp % base )
7        tmp =  tmp/base
8     }
9     nuevo.unshift( tmp )
10 }
```

En la línea 3 se crea el espacio que contendrá el nuevo número convertido a la base. En la línea 4 se inicializa la variable tmp con el número enviado (a ser convertido). El ciclo de 5 a 8 va agregando al inicio de la colección por el método del residuo el valor de la base convertida y luego divide el número hasta que el último residuo es menor a la base y es el primer elemento del número convertido (línea 9).

3.5.1 Algoritmo de composición de los grafos

A continuación se presenta la estructura abstracta de los grafos, es decir, cómo está formado computacionalmente el espacio de estados. Como se mencionó anteriormente, el espacio de estados está conformado por uno o más grafos, no hay relación entre ellos. Los grafos se componen a partir de la clase Nodo y las referencias que tienen hacia otros nodos, específicamente la variable hijo de tipo Nodo y la colección de objetos de tipo Nodo, padres, además de la variable root de tipo Nodo.

El algoritmo de pintado se encarga de crear las posiciones x y y de los nodos, de tal forma que al crearlos cumplan con las posiciones gráficas necesarias para no sobreponerse

sobre los ejes. La Tabla 3-5 describe las características que se tomaron en cuenta para dibujar los grafos sobre el plano.

Pintar cada grafo	Visualizar espacio de estados.
Por donde empezar a dibujar	Usuario.
Cómo resolver el ciclo	Crear el espacio de estados de forma gráfica en conjunto con sus herramientas para navegar y analizar.
Darles el espacio entre nodos	El documento generado por el servicio del caso de uso Generar espacio de estados debe estar bien formado y debe ser transferido con éxito.
Manejar los niveles	Escuchador en la interfaz gráfica que registra cuándo el archivo terminó de transferir.
Contemplar espacios vertical y horizontal	Se convierte el documento recibido a objetos del lenguaje computacional. Se crean las relaciones entre los objetos creados para después colocarlos gráficamente en su posición *x* y *y*. El espacio de estados debe verse todo junto y no en partes.
Posicionar las curvas	Se calculan las posiciones de las curvas y sus inflexiones con el fin de maximizar su vista y aprovechar el espacio.
Curvas optimizadas	Se calcula la posición de los nodos de tal forma que al posicionarlos se encuentren cerca unos de otros y no sea necesario cruzar líneas o extender su longitud innecesariamente.

Tabla 3-5 Características importantes para posicionar un elemento gráficamente

Anteriormente se mencionó que la estructura de datos es un grafo dirigido cíclico, con la propiedad de tener sólo un ciclo. Al tener diferentes objetos relacionados con padres e hijos, se pensó en dibujar las estructuras de datos como indican Gross y Yellen (2004) [16], representando a los vértices (nodos) del grafo por puntos (o figuras geométricas como círculos o rectángulos) y las aristas por curvas. El algoritmo de pintado tomó en cuenta estas propiedades para generar las posiciones. Sin embargo, al crear el algoritmo se originaron las siguientes dificultades:

- Encontrar la posición de cada nodo respecto a todos los demás, es decir, la posición de cada nodo es dependiente de la posición de los demás nodos buscando no colocar dos nodos en la misma posición y sobreponerlos.
- Un algoritmo que ajustara la posición de los nodos cada vez que se insertaba un nodo resultaba demasiado costoso en tiempo de procesamiento, pues el algoritmo tendría que recorrer todos los nodos el número veces del total de los nodos menos uno, en el peor de los casos.

Lo mejor era tomar un punto de referencia y de ahí recorrer cada nodo para posicionarlo y recorrer los nodos una sola vez. Anteriormente se mencionó que como grafo con un solo ciclo, se buscó una raíz, y en algunas situaciones se tomó en cuenta como árbol. Esto mejoró la percepción del grafo de un nivel abstracto a un nivel gráfico.

La relación entre los nodos es de cero o muchos padres para un hijo, por lo tanto, un hijo puede tener varios padres. Esta relación hace que al final del grafo se encuentre un solo nodo. Éste es el que se toma en cuenta como nodo raíz, pero el árbol y sus hojas, crecen hacia arriba con las siguientes propiedades:

- Al menos un ciclo por cada grafo.
- Si el nodo raíz tenía cero padres, él era su propio hijo.
- Por cada ancestro que tuviera el nodo raíz se aumentaba un nivel, es decir, si el nodo tenía cero padres era de nivel 1; si crecía sólo uno o varios padres hacia arriba era de nivel 2, y si alguno de esos padres tenía un padre era de nivel 3 y así sucesivamente.

Pintar el grafo partiendo del nodo raíz como base implicaba que cada vez que se insertara un nodo hacia arriba y se reajustara la posición de los demás nodos del grafo en caso de necesitar espacio horizontal, lo cual desencadenaba, como en el problema anterior, recorrer los demás nodos recursivamente hasta tener a todos en posición y hacer esto cada vez que se insertaba un nuevo nodo.

La solución propuesta recorre los nodos una sola vez y les asigna una posición única cumpliendo con minimizar los cruces, minimizar las inflexiones, el área, la minimización del largo y la simetría. La maximización de ángulos se resuelve con la aplicación que pinta los nodos (Genodus).

El problema de manejar el ancho de los hermanos al mismo nivel fue lo que motivó a crear el algoritmo en la forma en que se creó. Debido a que no se sabría qué tan ancho era un grupo de nodos que se encontraban al mismo nivel, se decidió hacer el algoritmo de la siguiente manera:

De un arreglo donde se encuentran todos los nodos, se toma uno por uno (excepto si ya está pintado) y a partir de éste continúa con el algoritmo de pintado del grafo.

"Analizar un algoritmo ha venido a ser la predicción de recursos que un algoritmo requiere. Ocasionalmente, recursos como memoria, ancho de banda de comunicación o hardware computacional son de importancia primaria, pero continuamente es el tiempo computacional el que queremos medir." [19]

Este algoritmo es eficiente sólo para el tipo de grafos que necesita generar, los cuales son grafos tipo árbol hacia arriba, con un ciclo y un nodo raíz artificial, y se dice que es eficiente porque al momento de asignar las posiciones se recorre sólo una vez cada nodo dando una complejidad lineal al algoritmo.

"El problema de analizar las propiedades combinatorias de un tipo dado de grafo geométrico naturalmente produce la pregunta de caracterización de esos grafos los cuales admiten un tipo dado del dibujo por líneas rectas. Esto, en cambio, induce a la investigación del diseño de algoritmos eficientes para procesar tal dibujo cuando uno exista. Aunque estas preguntas están lejos de ser resueltas en general, muchas respuestas parciales aparecen en la literatura." [16]

Para decir que un algoritmo es eficiente, se toma en cuenta el análisis del algoritmo y el tiempo que toma en su ejecución. Thomas H. Cormen dice que analizar un algoritmo ha venido a ser la predicción de recursos que un algoritmo requiere. Ocasionalmente, recursos como memoria, ancho de banda de comunicación o hardware computacional son de importancia primordial, pero continuamente es el tiempo computacional el que queremos medir [19]. En este caso se toma en cuenta el tiempo computacional con procesadores genéricos y se usa la notación-O ya que "se puede describir el tiempo de ejecución de un algoritmo puramente inspeccionando la estructura general del algoritmo".

A continuación se muestra la estructura general del algoritmo con una descripción de su función en el programa principal.

```
1 for(int i=0;i<total;i++){
2   n1=nodos.get(i);
3   n1.draw();
4 }
```

el código anterior indica que los nodos se recorrerán una vez y cada uno se le aplicará la función draw()

```
1 draw(){
2   if(!pintado && this==root){
3        dibujaGrafo();
4   }
5 }
```

La función draw() no hará nada si el nodo ya está pintado o si el nodo es raíz. Si fuera raíz, el siguiente algoritmo no se ejecutaría correctamente produciendo un "ciclo infinito", por eso se seleccionan nodos que no sean el nodo raíz.

```
1 dibujaGrafo(){
2   dibujaPadres();
3   dibujaNodo();
4   dibujaHermanos();
5 }
```

Si no está pintado y/o no es raíz ejecuta la funciones en ese orden: dibujar padres, dibujar el nodo actual, y dibujar a sus hermanos.

```
1  función dibujaPadres(){
2    for(int i=0;i<padres.size();i++){
3         if(padres.get(i)==root ||
4              padres.get(i).pintado)
5              continue;
6         padres.get(i).dibujaGrafo();
7    }
8  }
```

Dibujar padres recorrerá una sola vez cada nodo, mientras cumple con la recursión dibujaGrafo() seguida por dibujaPadres() hasta que no tiene padres.

```
1  dibujaNodo(){
2    if(pintado)return;
3    if(padres.size()==0){
4         this.xx=root.xoff;
5         root.xoff+=Gefundus.hgap;
6    }else{
7         double mayor=0;
8         for(int i=0;i<padres.size();i++){
9              if(padres.get(i)==root)
10                  continue;
11             mayor=(padres.get(i).xx>mayor)?
12                            padres.get(i).xx:mayor;
13        }
14        double menor=mayor;
15        for(int i=0;i<padres.size();i++){
16             if(padres.get(i)==root)
17                  continue;
18             menor=(padres.get(i).xx<menor)?
19                            padres.get(i).xx:menor;
20        }
21        this.xx=((mayor-menor)/2)+menor;
22   }
23   pintado=true;
24 }
```

Se recorren los padres de cada nodo para obtener su posición y generar la simetría. Esto aumenta la complejidad.

```
1 función dibujaHermanos(){
2       for(int i=0;i<hijo.padres.size();i++){
3             Nodo sibling=hijo.padres.get(i);
4             if(sibling.pintado || sibling==root)
5                   continue;
6             sibling.dibujaGrafo();
7       }
8 }
```

Una vez que pintó a sus padres, y el actual pinta a los hermanos y continua con la recursión.

Ningún nodo se recorre dos veces debido a la "bandera pintado" que evita ciclos y recorrido innecesarios. Esto sugeriría que la complejidad del algoritmo fuera lineal, pero cada vez que se posiciona un nodo recorre a sus padres para obtener la posición simétrica, lo cual da como resultado una complejidad $O(n^2)$.

El siguiente paso es generar el documento con los resultados para exponerlo como respuesta del servicio y ponerlo a disposición de quien realizó la llamada al servicio, para esto se crea un documento XML[8] con el fin de ser analizado por el cliente.

3.5.2 Algoritmo de creación de documento

El proceso de Gefundus termina con la creación del documento que contiene las posiciones y la estructura de los grafos en un documento estandarizado que pueda ser interpretado por algún servicio que lo consuma, lo interprete y genere graficamente la información. En este proyecto se ha creado Genodus.

[8] XML eXtended Markup Language es un lenguaje que está hecho de entidades que contienen datos formados por caracteres de datos o etiquetas. Las etiquetas codifican la descripción de almacenaje del documento y su estructura lógica. XML provee un mecanismo para imponer límites en el almacenaje o estructura lógica. [20]

Para que Genodus pueda consumir el servicio que ofrece Gefundus es necesario que se envíen los parámetros correctos. Estos son las funciones que regulan los grupos de genes, el número de genes por nodo y el número de estados por gen. Los parámetros se recibirán a través del documento de Java para peticiones web, Servlet, que se encargará de procesarlos en Gefundus para después procesar un documento de salida.

Los servicios web proveen medios estándar de interoperación entre diferentes aplicaciones de software, ejecutándose en una variedad de plataformas y/o marcos de trabajo. [20]

XML resuelve un requerimiento de tecnología clave que aparece en muchos lugares. Ofreciendo un estándar, flexible y un inherentemente extensible formato de datos, XML reduce significativamente la carga de despliegue de muchas tecnologías necesarias para asegurar el éxito de servicios web. [20]

Se eligió este formato debido a que es un formato de intercambio de datos muy usado junto con JSON (*Javascript Object Notation*) [21], además de que *Actionscript* (el lenguaje utilizado por Genodus) tiene librerías nativas en su API (*Application Programming Interface*) para procesar datos en documentos XML.

El algoritmo que produce el XML tiene la siguiente secuencia:

1. Crear el nodo raíz del XML.
2. Recorrer nodo por nodo y verificar si es raíz.
    a. Si es raíz, crear el nodo grafo enviando como atributos quién es el nodo raíz, el ancho del grafo y el tamaño del ciclo.
    b. Crear el grafo a partir de la raíz.

A continuación se muestra la estructura general del algoritmo que crea el documento XML:

```
1  String getXML(){
2    Nodo nod;
3    String xxml="<gefundus totalnodes='"+total+"'>";
4    for(int i=0;i<total;i++){
5         nod=nodos.get(i);
6         if(nod==nod.root){
7              xxml+="<graph root='"+nod.indice
8                   +"' width='"+nod.xmax
9                   +"' size='"+(nod.getSize()+1)
10                  +"' cyclelength='"
11                  +(nod.getCycleLength()+1)+"'  >";
12             xxml+=""+nod.getXMLGraph()+"";
13             xxml+="</graph>\n";
14             xxml+="\n";
15        }
16   }
17   xxml+="</gefundus>";
18   return xxml;
19 }
```

A continuación se presenta el código que genera el documento XML en su totalidad, utilizando la función que se presentó en el código anterior.

```
1  String toXML(){
2    String str="";
3    str+="<node ";
4    str+="id='"+this.indice+"' ";
5    str+="value='"+this.valor.join(",")+"' ";
6    str+="child='"+this.hijo.valor.join(",")+"' ";
7    str+="level='"+nivel+"' ";
8    str+="x='"+Math.round(xx)+"' ";
9    str+="y='"+Math.round(nivel*Gefundus.vgap)+"' ";
10   str+=" />";
11   return str;
12 }
```

Es importante resaltar la creación de los atributos en los nodos para poder entender el documento.

Para <gefundus>:

totalnodes: es un número único del total de nodos en toda la aplicación.

Para <graph>:

root: es el nodo que se tomará como raíz artificial para el conjunto de nodos dentro del grafo.

width: es el ancho simulado en pixeles que ocupará el grafo horizontalmente.

size: es el número total de nodos dentro del grafo.

cyclelength: es el número de nodos que se encuentran dentro del ciclo.

Para <node>:

id: es el valor único del nodo dentro de toda la aplicación.

value: es el valor del nodo separado por comas. Se pensó en la separación por comas en caso de que se seleccione un valor en el número de estados posibles mayor a 10, de forma que pudieran diferenciarse los valores del nodo.

child: es el valor del nodo hijo separado por comas.

level: es el nivel al que se encuentra el nodo verticalmente.

x: es la posición en el plano superficial horizontalmente.

y: es la posición en el plano superficial verticalmente.

Al procesar los algoritmos anteriores en respuesta a la petición del cliente se genera un documento como el que se presenta a continuación. Para este ejemplo, se generaron

cuatro nodos y dos grafos. El primer grafo tiene como raíz el nodo con *id* igual a 2, tiene un ancho de 0, un tamaño de 3 nodos y tiene 2 nodos en el ciclo. Los nodos que pertenecen al grafo son el nodo 0, el nodo 1 y el nodo 2. El nodo 0 tiene los valores 0,0, está en el nivel 0, en las coordenadas 0,0 y su hijo es el nodo 0,1, que está en el nivel 1, aumentado 100 en la posición *y*, con su hijo 1,0 en el nivel 2 que aumentó a 200 en el eje *y*, y aquí es donde se nota el ciclo ya que el hijo de éste es 0,1 (su padre). Para el segundo grafo sólo se tiene un nodo con un ciclo hacia él mismo.

```
<?xml version="1.0" encoding="UTF-8"?>
<gefundus totalnodes="4">
     <graph root="2" width="0.0" size="3"
       cyclelength="2">
          <node id="2" value="1,0" child="0,1"
            level="2" x="0" y="200"/>
          <node id="1" value="0,1" child="1,0"
            level="1" x="0" y="100"/>
          <node id="0" value="0,0" child="0,1"
            level="0" x="0" y="0"/>
     </graph>
     <graph root="3" width="0.0" size="1"
       cyclelength="1">
          <node id="3" value="1,1" child="1,1"
            level="0" x="0" y="0"/>
     </graph>
</gefundus>
```

El documento que se genera tiene siempre el mismo formato de nodos y atributos; a este formato se le llama esquema XML (XML Schema) y este proyecto se utiliza únicamente para descripción del documento.

El esquema XML que se presenta a continuación muestra la composición inicial del nodo <gefundus> con un atributo "totalnodes" de tipo entero. Compuesto por elementos requeridos sin un orden en específico llamados <graph>, donde cada uno representa un grafo

procesado con los atributos que definen qué nodo es el nodo raíz, cuál es el ancho del grafo, el número de nodos que lo componen y el número de nodos que componen el ciclo del grafo.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="gefundus">
   <xsd:attribute name="totalnodes" type="xsd:integer"/>
   <xs:complexType>
      <xs:all>
         <xs:element name="graph">
            <xsd:attribute name="root" type="xsd:integer"/>
            <xsd:attribute name="width"
                           type="xsd:integer"/>
            <xsd:attribute name="size" type="xsd:integer"/>
            <xsd:attribute name="cyclelength"
                           type="xsd:integer"/>
            <xs:complexType>
            <xs:all>
               <xs:element name="node">
                  <xsd:attribute name="id" type="xsd:integer"/>
                  <xsd:attribute name="value" type="xsd:string"/>
                  <xsd:attribute name="child" type="xsd:string"/>
                  <xsd:attribute name="level" type="xsd:integer"/>
                  <xsd:attribute name="x" type="xsd:integer"/>
                  <xsd:attribute name="y" type="xsd:integer"/>
               </xs:element>
            </xs:all>
            </xs:complexType>
         </xs:element>
      </xs:all>
   </xs:complexType>
</xs:element>
</xs:schema>
```

El documento generado se produce como un documento XML versión 1.0 al servicio que lo llama. La razón por la cual se decidió calcular las posiciones x y y antes de enviarla a la

aplicación de representación gráfica (Genodus) es por cumplir el objetivo de optimizar los procesamientos, en el que Gefundus, en Java, procesará los números y se dedicará completamente a encontrar lógicamente las posiciones y Genodus se encargará únicamente de posicionar gráficamente los elementos, optimizando el tiempo de ejecución de la interfaz.

3.6 Interfaz gráfica

Modelar software es esencialmente un proceso de manipular entidades intangibles. Éstas eran típicamente recursos no físicos que residen en las mentes de las personas o aparecen en papel. Un modelo efectivo de proceso debe ser tan visual como sea posible, habilitando a los negocios y el personal de tecnología a ver los elementos de software como si fueran recursos concretos.

El desarrollo de la interfaz implicó tener siempre presente su objetivo: cómo se deben comportar los elementos de tal forma que el usuario se sienta cómodo con su uso y el sistema responda adecuadamente a las peticiones que se le hacen.

"El diseño de la interfaz describe la estructura y organización de la interfaz del usuario. Incluye una representación de la plantilla de la pantalla, una definición de los modos de interacción y una descripción de los mecanismos de navegación." [15]

La aplicación Genodus se ejecuta en un ambiente de usuario sobre algún navegador de internet y *Flash Player*. *Flex* provee un conjunto rico de contenedores y controles que se usan para construir una interfaz de usuario. Un contenedor provee una estructura jerárquica para organizar y planificar la interfaz de usuario. Dentro de un contenedor se acomodan otros contenedores, navegadores, controles y otros componentes. [22]

La plantilla principal de la interfaz se construyó sobre *Flash Builder* en MXML, mezclando el código con *Actionscript* 3.0 con los siguientes elementos:

- MXML como lenguaje global.
- <fx:Script> es la sección con la que se comunica MXML y *Actionscript* 3.0.
- <fx:Style> es la sección donde se definen las clases de las hojas de estilo CSS.

Los contenedores se organizaron de tal forma que permitieran la fluidez y rigidez en los espacios necesarios para conservar los tamaños y así minimizar el espacio o maximizarlo donde se requería.

La Figura 3.13 muestra cómo están acomodados los contenedores unos dentro de otros para facilitar la plantilla de navegación.

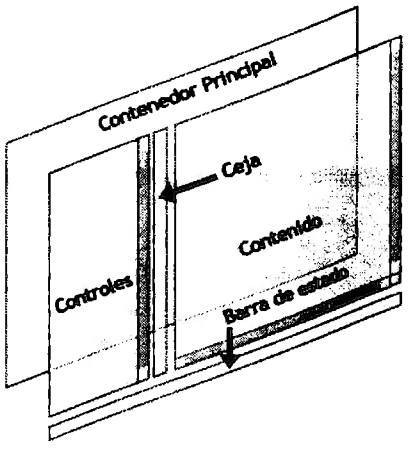


Figura 3.13 Organización de la interfaz en *Flex*

La interfaz (Figura 3.21) se desarrolló en idioma inglés, considerando que la mayoría de las personas en el mundo se puede comunicar en este idioma [23], y con la intención de aprovechar el uso vía internet de diferentes usuarios alrededor del planeta. Del lado izquierdo, en el contenedor con *id* "ceja" se encuentran los controles del usuario, donde debe escribir los parámetros de entrada para la aplicación.

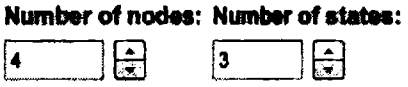


Figura 3.14 Parámetros de entrada de Genodus

En la Figura 3.14 el número de nodos es lo primero que el usuario puede escribir. Para esto se muestra una caja de texto donde puede aumentar o disminuir uno por uno el número de genes, presionando sobre los botones con flechas que se encuentran a su derecha. La caja de texto está restringida para que el usuario sólo pueda introducir números del 1 al 10. Esto es porque el sistema puede funcionar por lo menos con un elemento por nodo y un máximo de 10.

El número de estados por nodo es lo segundo que el usuario puede escribir. Para esto se muestra una caja de texto donde puede aumentar o disminuir uno por uno el número de genes, presionando sobre los botones con flechas que se encuentran a su derecha. La caja de texto está restringida para que el usuario sólo pueda introducir números naturales.

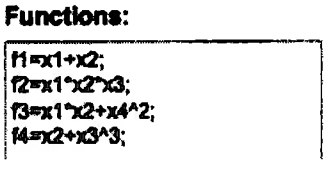


Figura 3.15 Funciones de entrada en Genodus

Los siguientes parámetros que deben introducir los usuarios son las funciones (Figura 3.15). Para introducir las funciones se colocó una caja de texto donde el usuario puede pegar las funciones desde un archivo de texto o escribirlas directamente en ella.

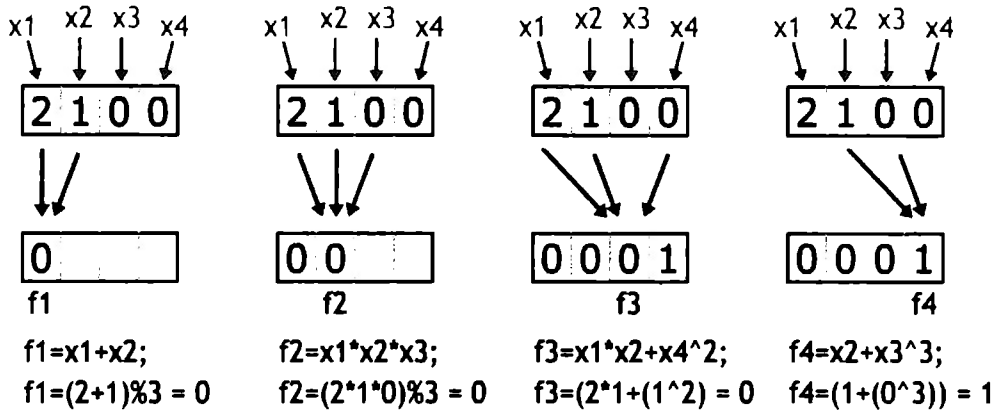


Figura 3.16 Regulación entre nodos por medio de funciones

Las funciones tienen la tarea de crear la interacción entre nodos, obteniendo de ellos los valores del nodo actual en la posición descrita por la variable X (x1, x2, etc.) y posicionando el resultado en la posición descrita por la variable F (f1, f2, etc.), dando así como resultado un nuevo nodo. Por lo tanto, debe existir el mismo número de funciones que el número de genes por nodo. La Figura 3.16 muestra un ejemplo de cómo se crea un nuevo nodo a partir de las funciones. Con la notación disponible actualmente, las semánticas de un lenguaje son mucho más difíciles de describir que la sintaxis. Para especificar semánticas debemos entonces usar descripciones informales y ejemplos sugestivos [17].

Las funciones que debe introducir el usuario manejan un lenguaje común para poder ser interpretado por la aplicación en este caso, se maneja un lenguaje matemático donde se coloca del lado izquierdo una función, un símbolo de igualdad, y la expresión aritmética que produce el resultado. El lenguaje que se maneja dentro de la aplicación debe ser definido, analizado, interpretado y ejecutado de tal forma que se cumpla el objetivo de ejecutar las funciones que producen la relación entre nodos. Lo anterior debe poder hacerlo el programa simplemente introduciendo el flujo de datos como entrada.

Este comportamiento es similar a escribir un lenguaje computacional, compilarlo y ejecutarlo, es por esto que se siguió la teoría de compiladores. Alfred Aho establece que los

lenguajes de programación son notaciones para describir computaciones a personas y a máquinas. El mundo como lo conocemos depende de los lenguajes de programación, porque todo el software que corre en las computadoras se escribió en algún lenguaje de programación. Pero, antes de que un programa se pueda ejecutar, primero debe ser traducido a una forma en la que pueda ser ejecutado por una computadora. A esto se le llama compilador. Asimismo, el lenguaje que introducirá el usuario son funciones a ser ejecutadas por la aplicación para relacionar los nodos, que debe ser traducida al lenguaje de programación Java para su ejecución a un nivel computacional.

Para que un lenguaje pueda ser analizado y ejecutado, Alfred Aho [17] define tres partes principales:

1. Analizador léxico. Su propósito es leer el flujo de caracteres, carácter por carácter, creando grupos de caracteres con un significado, llamados lexemas.
2. Analizador sintáctico. También llamado Parser, usa los lexemas para crear un tipo de árbol intermedio de representación que representa la estructura gramatical del flujo de lexemas.
3. Analizador semántico. Usa la sintaxis de árbol y el valor de los lexemas para hacer consistencia significativa entre lo escrito y la lógica de una definición de lenguaje.

3.6.1 Analizador sintáctico y semántico

El analizador sintáctico y semántico presentado en este proyecto es una solución a la medida sobre el lenguaje que debe acepta el lenguaje y lo ejecuta. El analizador está basado en el código de la sección 13.5 Recursiones Mutuas [24] con las modificaciones necesarias para aceptar el lenguaje específico para esta aplicación.

La solución de analizador que se presenta en este proyecto no sigue las reglas convencionales de un compilador, ya que no crea un árbol sintáctico ni crea una tabla de símbolos. Al ser una gramática que debe evaluar expresiones aritméticas, realiza lo que Alfred Aho llama "generación de código intermedio" donde evaluaciones toman lugar entre el análisis sintáctico y semántico y dan un resultado en el proceso:

Los pasos del algoritmo que generan el resultado del análisis son los siguientes:

1. Se recorre nodo por nodo.
2. Por cada nodo, se recorre cada función y se agrega el resultado en la posición correspondiente.

El código del analizador de cada función es el siguiente:

```
1  private Nodo n;
2  private ExpressionTokenizer tokenizer;
3  public Evaluator(Nodo n,String anExpression){
4    this.n=n;
5    tokenizer=new ExpressionTokenizer(anExpression);
6  }
```

Se crea un *tokenizador* encargado de revisar los caracteres, los siguientes caracteres y el nodo original.

```
1  public int getExpressionValue(){
2    int value = getTermValue(); boolean done = false;
3    while(!done){
4         String next = tokenizer.peekToken();
5         if("+".equals(next) || "-".equals(next)){
6              tokenizer.nextToken();//Discard "+" or "-"
7              int value2=getTermValue();
8              if("+".equals(next)) value=value+value2;
9              else value = value-value2;
10        } else done=true;
11   } return value; }
```

Se obtiene la expresión para obtener los términos con suma o resta y evalúa recursivamente:

```
1  private int getTermValue(){
2    int value = getPowerValue();
3    boolean done=false;
4    while(!done){
5         String next = tokenizer.peekToken();
6         if("*".equals(next) || "/".equals(next)){
7              tokenizer.nextToken();
8              int value2=getPowerValue();
9              if("*".equals(next))
10                  value=value*value2;
11             else
12                  value=value/value2;
13        }
14        else done=true;
15   }
16   return value;
17 }
```

Una vez evaluada, se obtienen los términos con multiplicación o división, y lo multiplica por el valor guardado "value2".

```
1  private int getPowerValue(){
2    int value = getFactorValue(); boolean done=false;
3    while(!done){
4         String next = tokenizer.peekToken();
5         if("^".equals(next)){
6              tokenizer.nextToken();
7              int value2 = getFactorValue();
8              if("^".equals(next))
9                   value = (int)Math.pow((double)value,
10                  (double)value2);//value=value*value2;
11        }
12        else done=true;
13   }
14   return value;
15 }
```

Dentro de ésta, se evalúa recursivamente para obtener las funciones de potencia y busca el número al cual quiere elevarse la potencia. Este valor se almacena en "value2".

```
1  private int getFactorValue(){
2    int val;
3    String next=tokenizer.peekToken();
4    if("(".equals(next)){
5         tokenizer.nextToken(); //Discard "("
6         val=getExpressionValue();
7         tokenizer.nextToken(); //Discard ")"
8    }else{
9         String str=tokenizer.nextToken();
10        if(str.charAt(0)=='X' || str.charAt(0)=='x'){
11             int ind=Integer.parseInt(str.substring(1))-1;
12             val=n.valor.get(ind);
13        }else{
14             val=Integer.parseInt(str);
15        }
16   }
17   return val;
18 }
```

getFactorValue obtiene el valor de una variable o un número, o si contiene paréntesis evalúa lo que está dentro del paréntesis, haciendo recursión ejecutando estos algoritmos nuevamente en el mismo orden. Si no contiene paréntesis se evalúa si es una variable (x1, x2, etc.) o si es un entero. No se aceptan números flotantes ni negativos, sólo naturales.

La gramática que acepta esta aplicación es sencilla pero debe estar muy bien restringida para su correcta evaluación. Para especificar la sintaxis, se presenta una notación ampliamente usada, llamada gramática libre de contexto o forma Backus-Naur (BNF) [17].

Backus—Naur Form

```
<funciones> ::= <funcion>
                | <funcion> <funciones>

<funcion>   ::= <simbolof> <indice> "=" <expresion> ";" EOL
                | <simbolof> <indice> "=" <expresion> ";"

<expresion> ::= "(" <expresion> ")"
                | <expresion> "-" <expresion>
                | <expresion> "+" <expresion>
                | <expresion> "*" <expresion>
                | <expresion> "/" <expresion>
                | "(" <expresion> ")" "^" <number>
                | <equis> <índice> "^" <number>
                | <equis> <indice>

<simbolof>  ::= "f" | "F"

<equis>     ::= "x" | "X"

<indice>    ::= [0-9] | [0-9] <indice>
```

Aunque el proceso de analizar gramaticalmente y semánticamente las funciones lo realiza Gefundus, se mencionó en esta sección para resaltar la importancia de la gramática que debe aceptar la caja de texto que utiliza el usuario para escribir las funciones.

Figura 3.17 Botón de envío de parámetros y procesamiento de Genodus

Genodus está preparado para enviar los parámetros si se seleccionó un número de genes por nodo, un número de estados y el mismo número de funciones que el número de genes por nodo.

Al presionar el botón "Generate Graph" se ejecuta la función que prepara la interfaz para dibujar los nodos y después se crean las variables de tipo URL, que se enviarán al servicio web Gefundus, localizado en la dirección URL http://marcos-bernal.appspot.com/gefundus en el puerto 80 o https://marcos-bernal.appspot.com/gefundus en el puerto seguro 443.

Para llamar al servicio es necesario enviar tres variables y sus valores: *functions*, *states* y *nodes*, donde *functions* define las funciones a ser procesadas, *states* define el número de estados y *nodes* define el número de genes por nodo.

Las variables son creadas para ser enviadas por tipo POST pero también pueden ser enviadas por GET siempre y cuando en cualquiera de los envíos las variables y sus valores estén codificadas para URL, esto es, que tengan el tipo *application/x-www-form-urlencoded* o el formato llave-valor con posibles llaves duplicadas, separadas por el carácter '=' de tal forma que una para llamar al servicio Gefundus por tipo GET resulta en:

http://marcos-bernal.appspot.com/gefundus?functions=f1%3Dx1%2Bx2%3Bf2%3Dx1*x2*x3%3Bf3%3Dx1*x2%2Bx4^2%3Bf4%3Dx2%2Bx3^3%3B&states=3&nodes=4 que es equivalente a:

Número de elementos por nodo: 4

Número de estados: 3

Funciones:

$f1 = x1 + x2$;

$f2 = x1 * x2 * x3$;

$f3 = x1 * x2 + x4^2$;

$f4 = x2 + x3^3$;

Este tipo de tecnología envía el formulario al servlet de Gefundus, recibiendo los parámetros *nodes*, *states* y *functions*. El servlet verifica si los datos son enteros para nodes y states y crea una nueva instancia de Gefundus pidiendo que imprima el XML que genere con un formato de archivo *text/xml*; y una codificación de caracteres *charset=UTF-8*.

Para que se pueda ejecutar este servlet al momento de consultar la URL relativa "/gefundus" es necesario especificar al servidor de aplicaciones appspot.com por medio del archivo web.xml que cuando un usuario consulte la dirección URL deberá ejecutar el servlet Gefundus, que a su vez es equivalente a la clase mx.itesm.GefundusServlet quedando de la siguiente manera:

```
<servlet>
    <servlet-name>Gefundus</servlet-name>
    <servlet-class>mx.itesm.GefundusServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Gefundus</servlet-name>
    <url-pattern>/gefundus</url-pattern>
</servlet-mapping>
```

Gefundus termina su ejecución cuando imprime el archivo XML desde el servlet, el servicio que realiza la petición es el encargado de leer el documento y procesarlo. La tecnología de Flash utiliza la tecnología Rich Internet Applications. Una aplicación rica de internet (RIA) atrae usuarios, les permite cumplir con tareas y puede ser absolutamente disfrutable. Permite a los colegas colaborar simple y transparentemente, y a los consumidores interactuar con productos de maneras emocionantes e irresistibles. RIAs van más allá de la vistas estáticas de la web tradicional y ponen dinámicamente contenido a medida justo en las manos del usuario. Continuamente trascienden las limitantes percibidas por navegadores web y entregan experiencias que los usuarios esperan de sus aplicaciones de escritorio. [25]

Cuando Genodus recibe la respuesta, existen dos posibilidades: 1) el evento se completó con éxito, y 2) el evento tuvo un error en la lectura o escritura del documento, ya sea por un error en la comunicación, por error en la generación del documento o por otras razones.

Si al recibir la respuesta desde el servidor al que se consultó el servicio (Gefundus) se recibe un error, se presenta un mensaje al usuario donde sólo puede aceptar que hubo un error, y se le regresa a la ventana principal.

Si en cambio recibe el documento completamente, lo primero que se hace es interpretar el documento XML, leerlo y traducir el documento en estructuras de datos lógicas para Genodus, de tal forma que se generen las mismas estructuras de datos que Gefundus generó.

Crear las estructuras de datos sirve además para empezar a preparar el ambiente donde se dibujarán los nodos. Por lo que se obtiene el ancho de los grafos y se crea el "lienzo" donde se dibujarán los grafos con el ancho y alto máximos a ocupar.

En el momento en que Genodus termina de crear las estructuras de datos, también obtiene la información del documento XML acerca de cuántos nodos se procesaron, cuántos grafos existen, el tamaño del grafo en número de nodos, el número de nodos que conforman el ciclo en cada grafo, cómo se conforman los ciclos y el valor del nodo en caso de ser un punto fijo. El punto fijo representa un ciclo de un nodo, por lo tanto se muestra el valor de ese nodo.

Analysis of the states space:
Total: 3 ^ 4 = 81
There are 9 components and 6 fixed points

Components	Size	Cycle Length	Value
1	1	1	0,0,0,0
2	8	2	
3	6	1	0,0,1,1
4	3	1	1,0,0,0
5	13	2	
6	15	1	1,0,1,1
7	2	1	2,0,0,0
8	14	2	
9	19	1	2,0,1,1

Figura 3.18 Tabla de estadísticas en Genodus

La aplicación en este punto se encuentra lista para procesar la tarea que ocupa más recursos de la computadora, que es el representar gráficamente los nodos, sus relaciones y los ciclos de los estados estacionarios.

3.6.2 Dibujar Nodos

En este proyecto la generación de imágenes fue uno de los principales problemas, debido a que se procesaban las imágenes por separado y se descargaba una imagen fuera de un tamaño visual comprensible conteniendo a todos los subgrafos, o se tomaban en cuenta diferentes métodos de procesamiento separadamente como *grid computing* o se procesaban visualmente en el cliente.

Debido a que el procesamiento de imágenes consume el mayor tiempo de los recursos de este proyecto, se decidió crear un archivo XML con la información de posicionamiento para construir gráficamente con la información de posicionamiento y dimensiones previamente procesada. De esta forma baja el procesamiento del servidor y el

cliente se encarga de procesar las imágenes, y tiene la opción de cancelar la operación si el proceso toma mucho tiempo en procesarse.

Flash Player tiene la habilidad de dibujo vectorial. De esta forma, en lugar de crear una imagen con información pixel por pixel se crean objetos vectoriales, que *Flash Player* se encuentra optimizado para representar.

La decisión de dejar que el cliente procese los grafos gráficamente crea más oportunidad de problemas debido a que si el usuario procesa muchos nodos la computadora del cliente tomaría muchos recursos de procesamiento. [26]

Independientemente del tipo de procesamiento que se eligiese, el algoritmo de despliegue de imágenes, y el proceso de generar y desplegar las imágenes en pantalla, consumiría recursos del CPU, haciendo que los periféricos con menor prioridad de la computadora no respondan, generando una sensación de "bloqueo" de la computadora.

El que se bloqueara la aplicación sería un problema especialmente para un usuario que no desea esperar a que la aplicación termine de procesar. Es por eso que en este proyecto se buscó solucionar el procesamiento continuo de los nodos, sin bloquear el procesador o cualquier recurso que se esté utilizando en ese momento.

La solución que se propone es utilizar una programación basada en eventos. *Flash Player,* al ser un programa creado, entre otras cosas, para crear animaciones, tiene una función llamada *onEnterFrame* que es un evento que se dispara cada que existe una llamada a un nuevo cuadro en la ejecución de la escena principal.

Se apalancó la programación con esta función para crear la programación basada en eventos, donde cada que entre a un nuevo cuadro llamará una etapa en el algoritmo para dibujar. El proceso de dibujar un nodo agrega una caja de texto (de la clase *TextField*) al lienzo (de la clase *SpriteVisualElement*) en la posición indicada.

Una vez que se colocaron todos los nodos en el lienzo, lo siguiente es crear las líneas que representarán gráficamente la relación entre nodos. El proceso para dibujarlas sobre el lienzo es el mismo, programación por eventos. Lo complicado fue crear una curva que fuera hasta donde continua el ciclo. La posición, la forma y la dirección de la curva fueron factores que determinaron la solución propuesta.

Kauffman [27] enlista un conjunto de criterios estéticos de importancia práctica:

- **Minimizar los cruces**. Si hay muchos cruces, el ojo humano no puede encontrar fácilmente cuáles nodos están conectados a una línea.

- **Minimizar las inflexiones**. Éste es un criterio importante de estética para planos (layout) ortogonales, porque el ojo humano puede seguir más fácilmente una línea con pocas inflexiones.

- **Minimizar el área**. Minimizar el área del plano es crucial para la integración a gran escala (VLSI).

- **Maximizar los ángulos**. Si el grafo se muestra en un dispositivo con poca resolución, es importante que las líneas estén tan lejos unas de otras como sea posible.

- **Minimización del largo.** Al integrar en gran escala es importante que las líneas sean lo más cortas posible.

- **Simetría.** Si un grafo contiene información simétrica, entonces es importante reflejar la simetría en su plano. Desafortunadamente, representar las simetrías no es una tarea fácil.

3.6.3 Dibujar Curvas

El proceso de pintar la línea sólo involucra al nodo padre y al nodo hijo. Si se saben sus posiciones, sólo es cuestión de colocar la línea debajo del padre y en el centro, y arriba del hijo y en el centro.

La función que se encarga de pintar la línea también pinta todos los nodos de un mismo color. En el caso de los ciclos de un solo nodo, los pinta de color amarillo (Figura 3.19), y en el caso de un ciclo de más de un solo nodo, los pinta de color verde claro, como se muestra en la Figura 3.20.

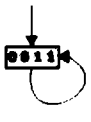


Figura 3.19 Ciclo de un solo nodo

El algoritmo que crea las líneas necesita dos puntos, el origen y el final. Como el comportamiento de las relaciones es similar en todos los grafos y en todos los nodos, se identificaron tres formas de líneas: las líneas que tienen una relación hacia abajo, las líneas que hacen un ciclo hacia el mismo nodo y las líneas que van hacia un nodo en una posición superior.

Maximizar los ángulos es algo importante en este tipo de grafos, debido a que no se sabe cuántos nodos puedan ser padres de un solo hijo, y representar la unión de esos padres en el hijo resultaba una tarea complicada.

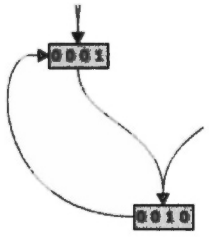


Figura 3.20 Ciclo de dos o más nodos

El pintar una línea hacia un nodo superior también complicaba un poco la tarea, debido a que era necesario calcular la posición del nodo superior tratando de no cruzar los nodos u otras líneas y sin cruzar la línea con las líneas que bajan de los padres buscando que la relación fuera clara visualmente.

La solución ideal supone una línea curva que se ajuste al modelo visual de los nodos. Es por lo anterior que se decidió crear líneas curvas utilizando polinomios cúbicos.

Los polinomios cúbicos son los más usados, debido a que polinomios de bajo grado dan muy poca flexibilidad en controlar la forma de la curva, y polinomios de más alto nivel pueden introducir resultados inesperados y también requerir más cálculo. Ninguna representación polinomial de grados bajos permiten a una curva la interpolación (paso a través) de dos puntos especificados con derivadas especificadas en cada punto final. Dado un polinomio cúbico con sus cuatro coeficientes, cuatro conocidas son usadas para resolver los coeficientes desconocidos.

Además de las ventajas que menciona Foley [28], se eligió una curva cúbica porque es la curva de más bajo nivel polinomial que permite dos inflexiones en la curva. Esto era

conveniente para hacer una curva del nodo al salir, flexionar la curva a la mitad del camino y flexionarla de nuevo para dirigirse al nodo final, haciendo de las curvas un camino más limpio y dando espacio entre ellas, maximizando los ángulos.

Para una línea que va del padre al hijo, se colocó como punto de control la posición del nodo origen en el eje horizontal y la mitad de la diferencia en la distancia respecto al nodo final en el eje vertical. Para el segundo punto de control se tomó la posición del nodo final en el eje horizontal y la mitad de la diferencia en la distancia respecto al nodo origen en el eje vertical. Para una línea hacia arriba se tomó como punto de control, la posición del nodo raíz al nivel vertical, y dependiendo de si el nodo hijo estaba a su derecha o a su izquierda se realizó un desfase de 40 pixeles en el eje horizontal. Para una línea en donde el destino es el mismo nodo, se buscó hacer una curva que pareciera un círculo, de forma que se notara un ciclo hacia el mismo nodo.

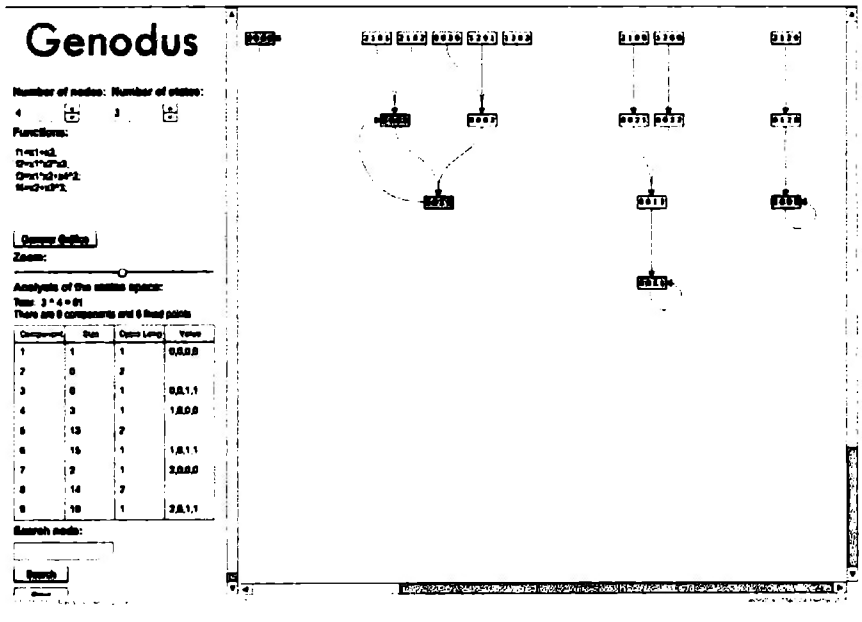


Figura 3.21 Espacio de estados creado por Genodus y Gefundus

La creación de la flecha simulando la relación entre nodos generaba el problema de no saber con qué ángulo orientar la cabeza de la flecha, es decir, el triángulo, especialmente en las

flechas que iban hacia arriba del nodo. Para simplificar el método de conocer el ángulo de las cabezas de las flechas, éstas se posicionaron para una línea de relación padre hijo normal, con la flecha hacia abajo. Para una línea que iba hacia arriba, se escogió el lado y se colocó completamente horizontal, al igual que las flechas que hacían un ciclo en el mismo nodo.

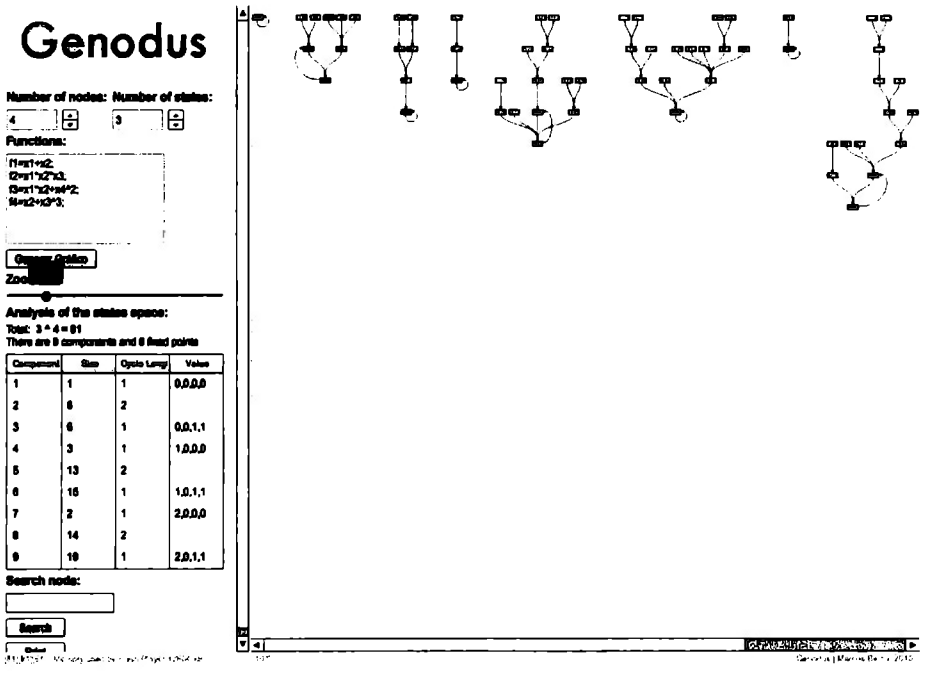


Figura 3.22 Herramienta de aumento y disminución de tamaño de Genodus

El espacio de estados es creado y se muestran grafos de diferentes tamaños y profundidades, es por eso que se colocaron barras de desplazamiento vertical y horizontal, para que el usuario pueda navegar hacia el grafo de su elección identificando su ciclo (Figura 3.21).

Aunado a lo anterior, se creó un control de magnificación (*zoom*) donde el usuario puede ver el plano en general a una escala superior de 1.8 veces su tamaño y a una escala inferior de 0.05 veces su tamaño con intervalos de 0.05 veces su tamaño. Para usar este control es necesario arrastrar el botón hacia adelante o hacia atrás mientras los grafos cambian de tamaño, como se muestra en la Figura 3.22.

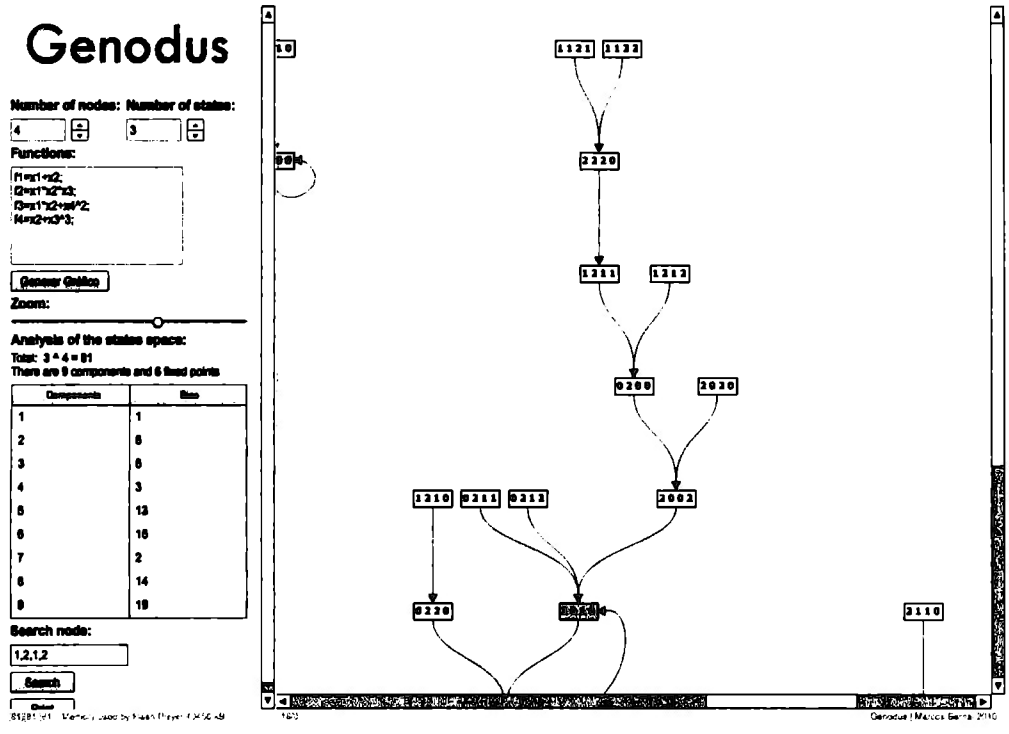


Figura 3.23 Herramienta de búsqueda de nodo de Genodus

Como parte de las herramientas que se ofrecen al usuario está la búsqueda de un nodo en particular. Para esto, es necesario escribir el valor de los estados del nodo separado por comas y presionar el botón con la etiqueta "Search". En caso de no encontrarlo, muestra un mensaje de error al usuario notificando que no se ha encontrado el nodo con el valor que ingresó en la caja de texto y es necesario modificar el texto. En caso de encontrarlo, el espacio donde están los grafos se moverá horizontalmente y verticalmente hasta encontrarlo y mostrarlo en el centro de este espacio remarcado con rojo para su fácil ubicación, como se muestra en la Figura 3.23.

La barra "plegable" es otra característica de Genodus. Esta ceja permite ampliar el espacio de visualización de los grafo, donde se presentan más cantidad de nodos y por consiguiente más análisis. El tamaño máximo depende de la resolución de la pantalla del usuario y del tamaño del navegador. El tamaño mínimo está definido por las dimensiones de los componentes, aproximadamente 400x160 pixeles. Además tiene un movimiento de

suavizado que ayuda a tener claro qué sucede con la parte que se mueve y adónde acudir en caso de necesitarla (Figura 3.24).

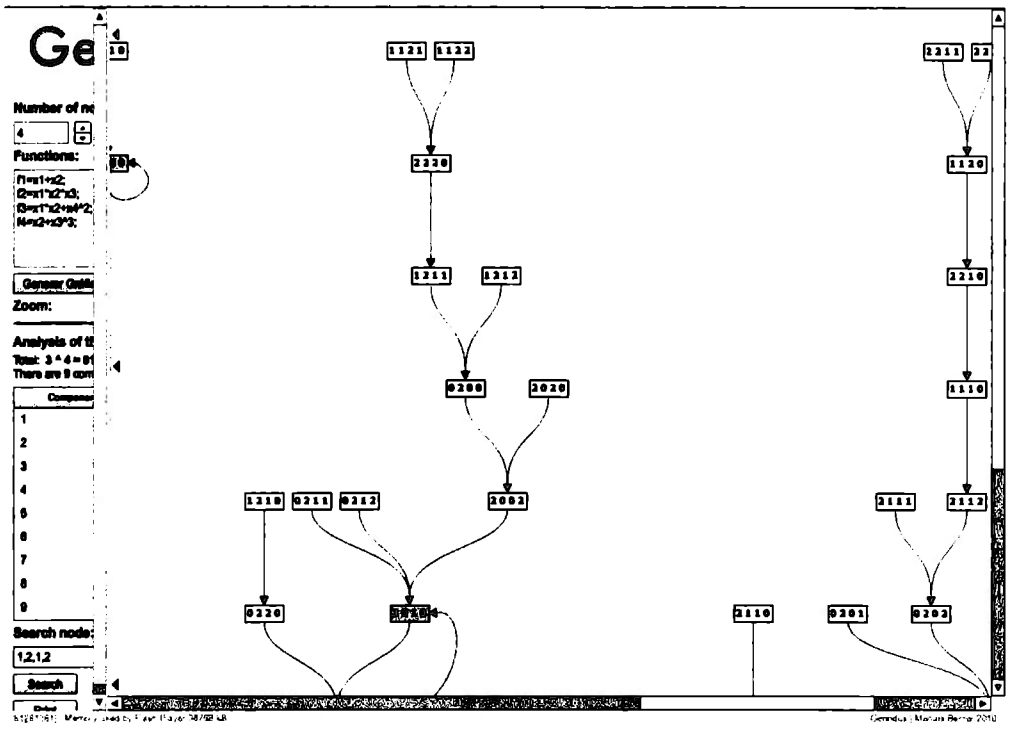


Figura 3.24 Barra plegable para maximización de espacio visual de Genodus

A veces los usuarios no desean analizar los grafos en una pantalla brillante, por lo que se incluye la función de imprimir los nodos. Esta función opera con las impresoras que el usuario tenga configuradas, pues se consulta directamente por las funciones que ofrece Flex en su API para imprimir. Sin embargo, hay algunas complicaciones que se presentaron, como son el tamaño del contenido de los nodos convertido en hojas y el corte de algunos grafos y líneas en los bordes de las hojas. El API de *Flex* y las configuraciones de márgenes de las impresoras son independientes de la aplicación, por lo que el usuario es responsable de esta configuración. Otra de las dificultades es que para grafos con muchos elementos normalmente no alcanza una hoja para colocarlo ya sea vertical u horizontalmente, por eso Genodus corta la imagen hasta donde quepa en la hoja y continúa en la siguiente hoja.

3.6.3.1 Memoria

La memoria de la computadora se vuelve un punto a considerar en el caso de que se quieran generar demasiados nodos. La cantidad de nodos a procesar por la aplicación debe ser limitada a los recursos que se tengan disponibles.

Gefundus utiliza la menor cantidad de objetos anidados y creados con el fin de optimizar la cantidad de bytes que ocupa cada objeto. Aun con esta metodología es importante cuidar la memoria que se está utilizando. Genodus tiene una barra en la parte inferior donde muestra cuántos nodos se han generado hasta el momento de forma lógica, cuántos nodos se han dibujado hasta el momento y cuántas líneas, además de la memoria en kilobytes (Kb) que utiliza en conjunto *Flash Player* como aplicación en el sistema operativo que se esté utilizando.

Debido a que se quieren aprovechar los recursos computacionales al máximo, y al hecho de no saber cuánta memoria va a utilizar la aplicación con una cantidad de nodos, así como el uso que le dé el usuario a su sistema operativo y memoria, se coloca una advertencia en el tutorial que se deben leer antes de utilizar Genodus.

3.6.3.2 Tutorial

Como parte de la publicación en internet, se incluye un tutorial donde el usuario puede aprender más sobre la intención de los proyectos y cómo usarlos. Se presenta una introducción (Figura 3.25) donde el usuario obtiene una idea general del resultado que se obtendrá.

Introduction

Genodus and Gefundus are tools that help visualize gene regulatory networks with multi-state discrete models. The aim of the software is to produce graphs like the one in Figure 1.

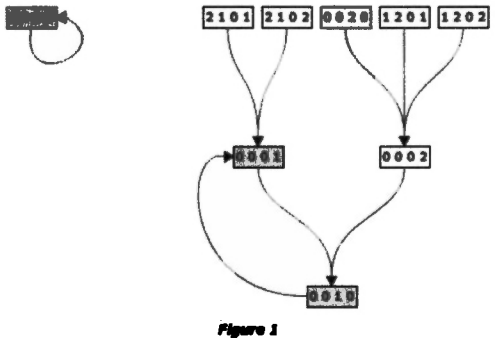


Figure 1

Figura 3.25 Tutorial de Genodus, Introducción

Además, se explica qué significa cada entrada del usuario, qué son los elementos por nodo, y cómo se representan en el modelo y su función (Figura 3.26).

Number of items per node (n)

The number of items per node represent the number of elements to be contained in a single node, the minimum number accepted by the software is 1 and the maximum is 10

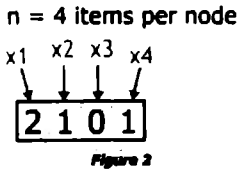


Figure 2

Figura 3.26 Tutorial de Genodus, Número de elementos por nodo

Se explica además la función de los estados por elemento (Figura 3.27), donde cada elemento puede tomar un valor dentro del rango dado y además se puede delimitar el valor máximo que pueda tomar.

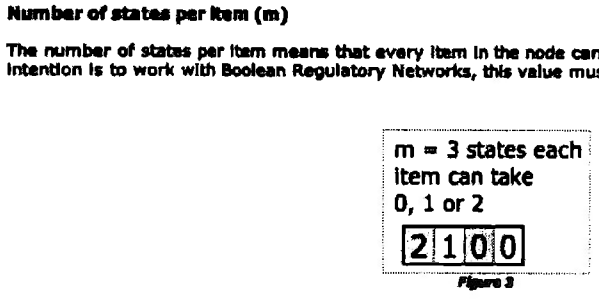


Figura 3.27 Tutorial de Genodus, Número de estados

Las funciones son algo que se debe explicar con cuidado, debido a que es un campo abierto en el que el usuario puede introducir cualquier conjunto de letras que podrían generar errores en la ejecución. Además, son la parte esencial para la regulación de la red. Se debe explicar el para qué se escriben las funciones, es decir, su principal objetivo dentro del sistema, y cómo actúan para la regulación en la red, de tal forma que cuando el usuario ingrese las funciones se pueda ejecutar con éxito el sistema. Además es necesario especificar la sintaxis y la forma de combinar caracteres para crear las funciones específicas. En la Figura 3.28 se presentan las funciones y cómo es que son procesadas, evaluadas y el significado de éstas.

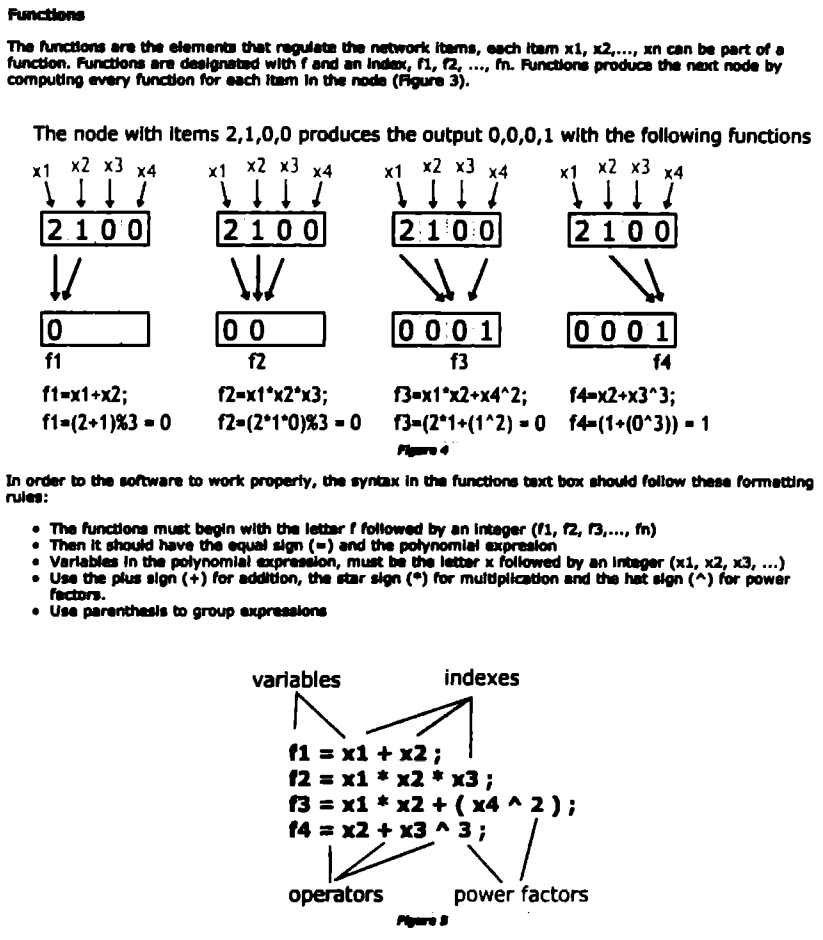


Figura 3.28 Tutorial de Genodus, Funciones de entrada

En este capítulo se presentaron los elementos que conforman el sistema, cómo se construyó y cómo funciona. La siguiente etapa en el desarrollo de esta aplicación web después de la planeación, el análisis, el diseño y la construcción es la etapa de pruebas.

La calidad de una WebApp –definida en términos de facilidad de uso, funcionalidad, confiabilidad, eficiencia, facilidad de mantenimiento, seguridad, escalabilidad y tiempo en el mercado– se introduce durante el diseño. Para lograr dichos atributos de calidad, un buen diseño WebApp debe poseer simplicidad, consistencia, identidad, robustez, navegabilidad y apariencia visual. [15]

CAPÍTULO 4
Validaciones

Las validaciones y pruebas son los procesos de ejercitar el funcionamiento o desempeño del software con la finalidad de identificar posibles errores o fallas. Esta filosofía fundamental no cambia para las WebApps. De hecho, puesto que los sistemas y aplicaciones basados en web residen en una red e interoperan con muchos sistemas operativos diferentes, navegadores (u otros dispositivos de interfaz como PDA o teléfonos celulares), plataformas de hardware, protocolos de comunicaciones y aplicaciones externas, la búsqueda de errores representa un desafío significativo para los ingenieros web. [15]

Las pruebas del sistema se realizaron continuamente y en diversas áreas del sistema con la intención de definir principalmente que el sistema cumple con los requisitos y los objetivos planteados al momento de la compilación de los requisitos.

Para realizar pruebas, de acuerdo con Alan Page [29], es necesario tener una estrategia. Un buen lugar para empezar es revisando los requerimientos y especificaciones funcionales. Después, preguntar cómo debería funcionar el sistema, cómo maneja los datos y cómo maneja los errores.

Principalmente se buscó hacer pruebas funcionales del siguiente tipo:

- La herramienta debe procesar las entradas como las funciones.
- Debe dar como resultado un espacio de estados donde se resalten visualmente los estados estacionarios (ciclos de cada nodo).
- Debe proveer una forma de visualización del espacio de estados y facilitar su uso.

4.1 Pruebas Funcionales

Para probar que la herramienta procesa las entradas correctamente se probó introduciendo diferentes valores posibles en cada uno de los parámetros de entrada. Si se procesó correctamente, se mostraría el espacio de estados visualmente con los estados estacionarios resaltados en diferentes colores.

4.1.1 Pruebas en condiciones normales

Probar en condiciones normales es la parte de las pruebas donde se ponen valores reales al azar para intentar cubrir el tipo de combinaciones que el usuario pondrá como entrada. Para este tipo de pruebas se eligieron valores diferentes para cada tipo de entrada, desde diferentes números de elementos por nodo hasta un número de estados incremental por cada cambio en el número de elementos por nodo, y un cambio en las funciones de modo que regulen en forma diferente cada cinco pruebas, con el fin de probar la variabilidad en el número de elementos por nodo, en el número de estados y en las funciones. Para este tipo de pruebas se limitó a probar hasta 20 mil nodos.

Con este tipo de pruebas en condiciones normales se esperaba que el comportamiento fuera predecible en cuanto a los resultados. Esto quiere decir que se esperaba que el proyecto se comportara adecuadamente y que lograra procesar cualquier tipo de combinación al azar al que se le sometiera.

4.1.2 Pruebas de límite

La idea de realizar las pruebas de límite según Alan Page [29] se debe a que algunos sistemas fallan en los límites de los parámetros, por lo que se prueba con los límites inferiores y los límites superiores, esto quiere decir que se prueba con los valores mínimos y los valores máximos. Además, se prueba con los valores mínimos menos uno, y máximo más uno, con la intención de tener una respuesta esperada de rechazo de los valores.

Se probó con cada variable al mínimo en cada parámetro de entrada obteniendo positivamente las respuestas esperadas. Se probó también cada variable al mínimo menos uno por cada parámetro. Directamente desde la interfaz, no permite la entrada a un valor menos uno, lo cual es un resultado de la prueba positivo. Sin embargo, para las funciones o para el paso de parámetros a Gefundus vía URL los parámetros pasan sin checarse al inicio, pero el sistema responde adecuadamente al marcar un error.

4.2 Pruebas no funcionales

Las pruebas no funcionales se basan en aquello para lo que el sistema no fue desarrollado específicamente que afecta la función del sistema. [29] En este sistema las pruebas no funcionales se basaron en los requisitos no funcionales de la Sección 3.2.2.

4.2.1 Rendimiento

Cada prueba se midió con un cronómetro en la mano, midiendo a partir de la acción de presionar el botón de "Generar grafo" hasta que se muestra el mensaje que indica que el espacio de estados se ha completado. Esto se realizó 10 veces y se sacó un promedio. La siguiente tabla representa los promedios en el tiempo.

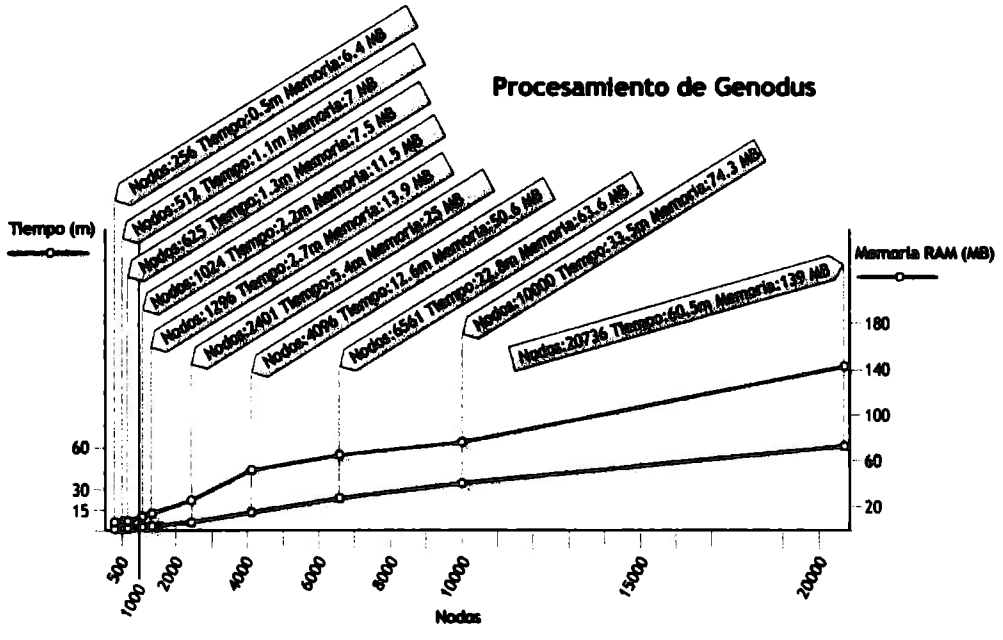


Figura 4.1 Tabla de desempeño de Genodus y Gefundus en memoria y tiempo de procesamiento

En la gráfica anterior se puede apreciar el comportamiento regular del tiempo en el procesamiento de los nodos conforme se aumenta el número de nodos. Esto puede indicar una posible predicción en el tiempo que tardará el sistema en producir cierto número de nodos.

4.2.2 Visualización

El diseño de la interfaz se guió con la idea de un espacio simple y claro donde el usuario pudiera observar fácilmente los lugares donde debe colocar las variables de entrada, así como las herramientas para manejar la información como zoom, tabla de estadísticas y búsqueda de nodo. Además, se buscó que los nodos se presentaran de una forma clara, con colores sencillos que lograran transmitir simplicidad, y así el usuario lograra enfocarse en el análisis.

4.2.3 Facilidad de navegación y usabilidad

La interfaz provee una interfaz clara y limpia, donde los elementos están acomodados en un espacio que permite su visualización entre ellos y además aprovecha el área de trabajo, excepto por los nodos en los grafos, que aunque aprovechan el espacio vertical, horizontalmente y por los algoritmos usados, se podría aprovechar de mejor manera el espacio horizontal. Esto se propone para un desarrollo futuro o mejora del sistema.

La interfaz además tiene claro los elementos que deben ser presionados (botones), los elementos que son de entrada y los elementos que son arrastrables, debido a que son elementos que se usan en otro tipo de sistemas con los que el usuario ya está familiarizado, como es un procesador de textos o el sistema operativo.

4.2.4 Multiplataforma y disponibilidad

La facilidad de visualizar el proyecto desde varias plataformas se debió principalmente al dispositivo y su versión de *Flash Player.* Se probó para los siguientes sistemas operativos con éxito: Windows 7, Windows XP, Ubuntu Maverick Meerkat (10.10), Android 2.3.4 Gingerbread y Android v4.01 Ice Cream Sandwich (Figura 4.2). Aunque la visualización depende del tipo de resolución del dispositivo del usuario, la prueba sobre multiplataforma fue exitosa, con excepción del tutorial y el navegador de los dispositivos móviles, que se espera mejoren en el futuro.

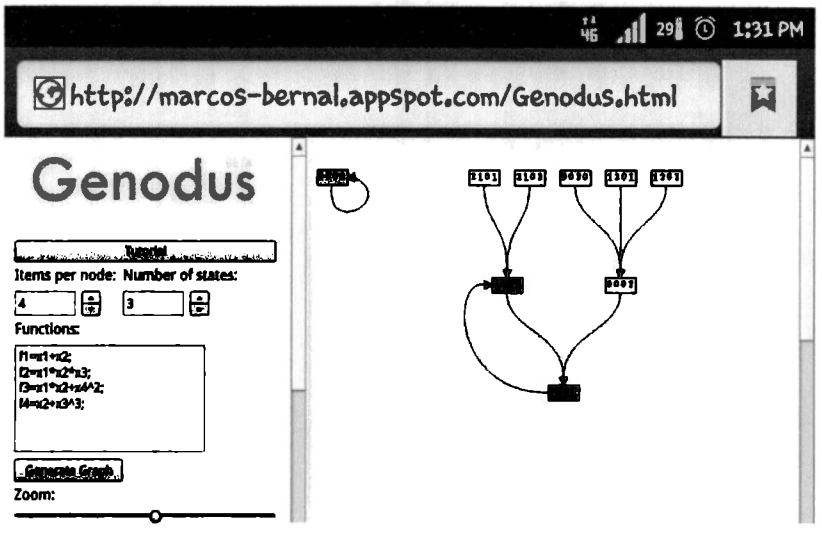


Figura 4.2 Genodus sobre un teléfono inteligente

4.3 Comparación con software existente y funciones similares

Como se mencionó anteriormente, los sistemas que se encarguen de representar los datos siguen creciendo. Sin embargo, muchos son especializados, como Genodus y Gefundus: el hecho de solo representar visualmente los espacios de estados y sus ciclos estacionarios, lo hace un sistema especializado.

El sistema más parecido y del que se tomaron ideas para realizar este sistema es el realizado por el Instituto de Bioinformática del Tecnológico de Virginia, el sistema llamado Visualizador Discreto de Dinámicas (DVD). Es por eso que se comparan las siguientes variables respecto a este sistema:

4.3.1 Función principal de los sistemas

Los dos sistemas se utilizan para visualizar redes biológicas de modelos discretos multi-estados y despliegan visualmente los grafos de los espacios de estados, resaltando los ciclos estacionarios con colores.

4.3.2 Parámetros de entrada

DVD utiliza campos de texto para introducir los parámetros "número de nodos" y "número estados por nodo". Además, utiliza una caja de texto para introducir las funciones y da la opción de cargar un archivo. Esta opción no está disponible en Genodus, pero puede desarrollarse como parte de la actualización de versión del proyecto. Además, DVD tiene especificación de estados por ver en caso de que no se quiera ver todo el espacio de estados generado.

4.3.3 Presentación de los resultados

DVD tiene opciones para representar los resultados en formatos *.gif, *.jpg, *.png y *.ps, mientras Genodus representa los datos en forma vectorial.

4.3.4 Rapidez de procesamiento

DVD produce resultados en un tiempo menor a un segundo, evaluado con un total de nodos menor a 1,000, mientras que Genodus produce resultados en un promedio de cinco nodos por segundo.

4.3.5 Límite de procesamiento

DVD tiene un límite de visualización de 1,000 nodos, mientras que Genodus tiene un límite de prueba de 20 mil 736, y se podría extender.

4.3.6 Construcción del sistema.

DVD está hecho con HTML y Perl, el cual no es aceptado por algunos navegadores. Genodus está hecho con *Flex* y HTML, el cual sólo es aceptado por los navegadores que tienen Flash Player instalado.

DVD utiliza principalmente el procesamiento de servidores administrados por el Tecnológico de Virginia, mientras que Genodus utiliza servicios gratuitos de hospedaje y dominio ofrecidos por Google. DVD deja la carga de procesamiento directamente en los servidores o en el cliente, descargando una aplicación con instalación especializada en Linux. Genodus procesa una parte en el servidor y le deja el procesamiento al cliente, dejando que éste se extienda hasta donde lo considere viable.

DVD utiliza una interfaz más estricta en el uso de botones y navegación debido a que hay que seleccionar varias opciones antes de procesar y, al procesar, además hay que desplazar la página hacia abajo, encontrar el enlace y presionarlo para ver la imagen. Si la imagen es muy grande, sólo se puede hacer una ampliación de cien por ciento incluida con el navegador, y utilizando las barras de desplazamiento del navegador. Genodus presenta todos los controles de un solo lado y sólo con presionar el botón de generar se puede observar cómo se genera el espacio de estados.

Genodus presenta herramientas de navegación y análisis, como son las barras de desplazamiento. El *zoom* cada cinco por ciento, desde cinco por ciento hasta 180 por ciento;

un buscador de nodo automático; controles plegables para una mejor apreciación del espacio de estados; impresión de los nodos y un tutorial gráfico.

Genodus puede ejecutarse en celulares y tabletas con navegadores que tengan *Flash Player*; DVD, sólo en navegadores que puedan interpretar Perl como página de internet.

Ambos presentan estadísticas del procesamiento. Genodus, además, presenta las estadísticas en una tabla con columnas que se pueden ordenar y filas que se pueden seleccionar.

4.4 Ejemplo con datos biológicos

Se incluye un ejemplo con el modelo del metabolismo de la lactosa en E coli (Michal 1999) [30].

Sea x1 glucosa

Sea x2 lactosa

Sea x3 β-galactosidase

Sea x4 el represor de la proteína

Para cada sustancia se toman en cuenta dos estados; 1 si está presente y 0 si no lo está.

Entonces las funciones de regulación son las siguientes:

$f1=x2*x3;$

$f2=x2;$

$f3=x2*(x4+1);$

$f4=1+x2;$

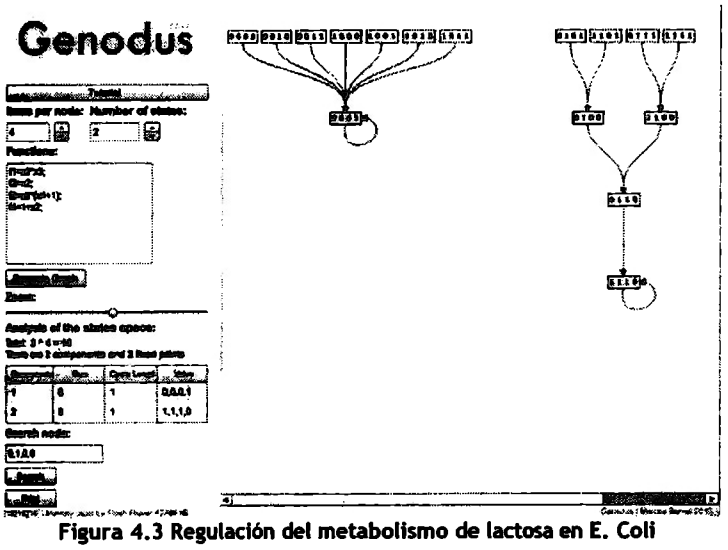


Figura 4.3 Regulación del metabolismo de lactosa en E. Coli

CAPÍTULO 5
Conclusiones

En esta sección se hablará sobre el alcance de los objetivos planteados, qué tanto se logró cubrirlos y hasta dónde se llegó. Se habla de las contribuciones y limitaciones del proyecto. Además, de las nuevas oportunidades y mejoras que pueden surgir de este proyecto.

Capacidad

Entre los principales objetivos se encuentran el procesar más de 1,000 nodos. Genodus y Gefundus trabajan en conjunto para procesar la red reguladora de hasta 20 mil 736 nodos mostrando gráficamente el espacio de estados y los ciclos estacionarios. Esto es más de 20 veces más el límite propuesto. El procesamiento de 20 mil 736 nodos es una de las contribuciones principales de este proyecto. Además, el sistema podría funcionar para el procesamiento de más nodos; sin embargo, no se ha asegurado su funcionamiento.

Tiempo

El tiempo de procesamiento de nodos de este proyecto es mayor que el del software DVD, y esto es debido a que Genodus crea el objeto para su manipulación gráfica, es decir, toma más tiempo porque tiene más funcionalidad.

El tiempo de generación de imágenes y manipulación gráfica es un aspecto que podría mejorarse a medida que las tecnologías web y de procesamiento gráfico vayan avanzando. Aunque a la fecha de creación del proyecto existe HTML5 como tecnología alternativa a *Flash Player*, no se realizó con HTML5 por las siguientes razones:

- Era necesario aprender el lenguaje y las nuevas tecnologías, lo cual retrasaría el tiempo de desarrollo del sistema, comparado con la experiencia del autor para crear el mismo sistema con *Actionscript* 3.0 y otras tecnologías conocidas.
- HTML5, al ser una tecnología emergente, podía ser propensa a errores desconocidos para la creación del sistema. Se pensó que lo más adecuado era ajustarse a una tecnología robusta, como es *Flash Player* en su versión 11.0, y a la experiencia del autor con el fin de enfocarse en la solución del problema en vez del uso de la tecnología emergente.

Es necesario resaltar que Genodus es una herramienta en línea que aprovecha los recursos del servidor y del cliente, por un lado, Gefundus procesa los grafos y sus posiciones, y por otro, Genodus siendo descargado desde el servidor hacia el cliente procesa la cantidad de nodos que sus recursos le permitan.

Extensión

Gefundus se ofrece como servicio web para que otras plataformas o sistemas que se desarrollen puedan utilizar el servicio de procesamiento y crear su propio sistema de visualización. La forma de consultarlo es directamente al servidor appspot.com de Google, lo cual puede tener limitantes en procesamiento. Sin embargo, el servicio se ofrece con crecimiento de procesamiento y otros recursos por una renta de acuerdo con la demanda. Esto se propone como un ajuste futuro al sistema en caso de requerir más recursos por su uso. Otra mejora es la generación de un documento JSON, debido a que es un documento más ligero y con fácil manejo desde Javascript.

Visualización

Genodus se desarrolló con el fin de procesar gráficamente los espacios de estado y poder manipularlos dinámicamente. Esto contribuye como un sistema que permite la visualización de espacios de estado con herramientas que facilitan la navegación y visualización al usuario. Dentro de los algoritmos de posicionamiento gráfico del grafo, se puede mejorar el aprovechamiento de espacio horizontal ya que, actualmente, cualquier nodo dentro de un grafo que no tenga padres ocupa un nuevo espacio en el eje horizontal, pero este espacio se podría mejorar para aprovechar el espacio.

Como parte de un mejor uso de la herramienta se provee de un tutorial gráfico que explica las partes funcionales del sistema. Es una contribución el tener explicaciones gráficas que ayuden a la comprensión de los procesos.

Se buscó la maximización del espacio de visualización con los controles plegables y así buscar aprovechar el espacio de visualización del usuario, independientemente de la resolución de su pantalla, especialmente en dispositivos móviles.

Se agregó la herramienta de búsqueda de nodo donde se presenta el nodo en la pantalla independientemente de la parte que se esté visualizando, y la herramienta de aumento y disminución de tamaño apoyando al objetivo de mejorar al análisis de los datos presentados. La búsqueda y el cambio de tamaño con la herramienta *zoom* cumplen con sus funciones y se complementan, debido a que cuando hay un aumento o disminución de tamaño la búsqueda ajusta la presentación del nodo intentando mostrarlo al centro de la pantalla. Sin embargo, cuando el tamaño es muy pequeño, la búsqueda del nodo no se aprecia. Una posible mejora puede ser el aumentar el tamaño a un tamaño visible cuando se realice una búsqueda.

La herramienta de impresión es algo que se incluye además en este sistema. Sin embargo, hay mejoras que hacer a la calidad de impresión debido a que cuando son grafos muy grandes se hacen cortes sin considerar los anchos o altos de los nodos en los grafos.

Los proyectos Genodus y Gefundus contribuyen a las herramientas disponibles en internet para el análisis de datos en una red reguladora genética con los siguientes puntos:

- Algoritmo de analizador sintáctico.
- Algoritmos de representación y recorrido del grafo.
- Representación gráfica de un grafo.
- Disponibilidad en internet.
- Interfaz gráfica con visualización dinámica.

Las mejoras y extensiones que se pueden agregar a estos proyectos son numerosas y aún queda un amplio camino en la investigación de redes reguladoras genéticas y la creación de herramientas disponibles para el análisis.

Con el desarrollo de más herramientas tecnológicas para el análisis de datos genómicos crece la visión para comprender nuestra esencia como seres vivos y la motivación por encontrar los patrones que conducen los procesos biológicos que nos hacen vivir y vivir mejor.

Bibliografía

[1] Ian Foster, *Designing and Building Parallel Programs*, 13th ed.: Addison-Wesley, 1995.

[2] Gil Alterovitz and Marco Ramoni, *Systems Bioinformatics*. Norwood, Massachusetts: Artech House, Inc., 2007.

[3] Neil Jones and Pavel Pevzner, *An introduction to bioinformatics algorithms*. Cambridge, Massachusetts: The MIT Press, 2004.

[4] Helen Claire Causton, John Quackenbush, and Alvis Brazma, *Microarray Gene Expression Data Analysis: A Beginner's Guide*. MA: Blackwell Publishing, 2003.

[5] Gregory Piatetsky-Shapiro and Pablo Tamayo, "Microarray Data Mining: Facing the Challenges," *ACM SIGKDD Explorations Newsletter*, pp. 1 - 5, 2003.

[6] M Madan Babu, "An Introduction to Microarray Data Analysis," in *Computational Genomics: Theory and Application*. Cambridge, UK: Horizon Bioscience, 2004, p. 306.

[8] Ilya Shmulevich, Ilya Gluhovsky, Ronaldo F. Hashimoto, Edward R. Dougherty, and Wei Zhang, "Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks," *Comparative and Functional Genomics*, 2003.

[7] María Alicia Aviñó, Edward Green, and Oscar Moreno, "Applications of finite fields to dynamical systems and reverse engineering problems," *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 191 - 196, 2004.

[9] Francesco Beltrame et al., "GEMMA - A Grid Environment for Microarray Management and Analysis in bone marrow stem cells experiments," *Future Generation Computer Systems*, pp. 382 - 390, 2007.

[10] C C Wu, H C Huang, H F Juan, and S T Chen, "GeneNetwork: An Interactive Tool For Reconstruction Of Genetic Network Using Microarray Data," 2003.

[11] Lev A Soinov, Maria A Krestyaninova, and Alvis Brazma, "Towards reconstruction of gene networks from expression data by supervised learning," *Genome Biology 2003*, 2003.

[12] Ian Foster, Carl Kesselman, and Steven Tuecke, "The Anatomy of the Grid, Enabling Scalable Virtual Organizations," *Intl J. Supercomputer Applications*, 2001.

[13] Hussein Vastani, Abdul Salam Jarrah, and Reinhard Laubenbacher, "Visualization of Dynamics for Biological Networks".

[14] Dr. Laubenbacher. (2009, Mayo) Applied Discrete Mathematics Group at VBI. [Online]. http://polymath.vbi.vt.edu/tiki/tiki-index.php

[15] Roger S. Pressman, *Ingeniería del Software. Un enfoque práctico*, 6th ed.: McGraw-Hill, 2005.

[16] Kenneth H. Rosen, *Handbook of Graph Theory*, 1st ed., Jonathan L. Gross and Jay Yellen, Eds. New York, United States of America: CRC Press, 2004.

[18] Judith Myerson. (2009, Mar.) developerWorks. [Online]. http://www.ibm.com/developerworks/web/library/wa-cloudgrid/

[17] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffery D. Ullman, *Compilers. Principles, Techniques, & Tools*, 2nd ed. New York, United States of America: Pearson Education, 2007.

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 1st ed. Cambridge, England: McGraw-Hill, 2001.

[20] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. (2000, Oct.) W3C recommendation. [Online]. http://www.w3.org/TR/2000/REC-xml-20001006

[21] Senthil Nathan, Edward J Pring, and John Morar. (2007, June) developerWorks. [Online]. http://www.ibm.com/developerworks/xml/library/x-xml2jsonphp/

[22] Adobe Systems Incorporated. (2011, January) Adobe Flex 4 Help. [Online]. http://help.adobe.com/en_US/Flex/4.0/AccessingData/WSbde04e3d3e6474c4-668f02f4120d422cf08-7ff4.html

[23] Seth Mydans, "Across cultures, English is the word," *The New York Times*, May 2007.

[24] Cay Horstmann, *Big Java*, 3rd ed. Hoboken, NJ, United States of America: John Wiley & Sons, Inc., 2008.

[25] Juan Sanchez and Andy McIntosh, *Creating Visual Experiences with Flex 3.0*, 1st ed. New York, United States of America: Addison-Wesley, 2009.

[26] Michael Kassoff, Daishi Kato, and Waqar Mohsin, "Creating GUIs for Web Services," *IEEE INTERNET COMPUTING*, September 2003.

[28] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics: Principles and Practice in C*, 2nd ed. United States of America: Addison-Wesley, 2004.

[27] Michael Kauffman and Dorothea Wagner, *Drawing Graphs. Methods and Models*, G. Goos, J. Hartmanis, and J. vanLeeuwen, Eds. Berlin, Germany: Springer, 2001.

[29] Alan Page, Ken Johnston, and Bj Rollison, *How we test Software at Microsoft*, 1st ed. California, United States of America: O'Reilly Media Inc., 2009.

[30] Gerhard Michal, *Biochemical Pathways*, 1st ed. New York, United States: Wiley-Spektrum.

[31] Google. (2010, January) Google App Engine. [Online]. http://code.google.com/appengine/docs/whatisgoogleappengine.html

[32] NVIDIA Corporation. (2012) What is GPU Computing? [Online]. http://www.nvidia.com/object/GPU_Computing.html